

Contents

[SQL Database Documentation](#)

[Overview](#)

[What is SQL Database?](#)

[Which SQL option should I choose?](#)

[Release notes](#)

[Release notes](#)

[Gateway IP address updates](#)

[Periodic maintenance events](#)

[Price and performance options](#)

[Purchasing models](#)

[Overview](#)

[vCore model](#)

[vCore model overview](#)

[Serverless](#)

[Azure Hybrid Benefit](#)

[Reserved capacity](#)

[DTU model overview](#)

[Migrate from DTU to vCore](#)

[Compute & storage](#)

[Overview](#)

[General purpose / Standard](#)

[Business critical / Premium](#)

[Hyperscale](#)

[Hyperscale FAQ](#)

[Migrate from Premium RS](#)

[Quickstarts](#)

[SQL databases](#)

[SQL managed instances](#)

[Concepts](#)

[Common features documentation](#)

[Feature comparison](#)

[How-to guide](#)

[Security](#)

[Security overview](#)

[Security best practices](#)

[Access management](#)

[Access control](#)

[Logins and users](#)

[Authentication](#)

[Configure Azure AD auth](#)

[Multi-factor AAD auth](#)

[Configure multi-factor auth](#)

[Data protection](#)

[Transparent Data Encryption \(TDE\)](#)

[TDE with Azure SQL](#)

[TDE with Bring Your Own Key](#)

[Configure TDE with BYOK](#)

[Rotate TDE BYOK keys](#)

[Remove TDE protector](#)

[Dynamic data masking](#)

[Always encrypted](#)

[Use the certificate store](#)

[Use the Azure key vault](#)

[Monitoring, logging, and auditing](#)

[Auditing](#)

[Get started with SQL Database auditing](#)

[Security management](#)

[Advanced data security](#)

[Data discovery and classification](#)

[Vulnerability assessment](#)

[Advanced Threat Protection](#)

- [Security attributes](#)
- [Single database security](#)
- [Network access controls](#)
- [IP-based firewall](#)
- [vNet firewall rules](#)
 - [vNet endpoints](#)
 - [vNet endpoints - PowerShell](#)
- [Private Link](#)
- [Audit Log Format](#)
- [Conditional Access](#)
- [Managed instance security](#)
 - [Secure public endpoints](#)
 - [Managed instance auditing](#)
- [Security baseline](#)
- [Connect and query](#)
 - [Connect and run ad-hoc queries](#)
 - [Azure Data Studio](#)
 - [SSMS](#)
 - [Azure portal](#)
 - [VS Code](#)
 - [Connect and query from apps](#)
 - [.NET with Visual Studio](#)
 - [.NET Core](#)
 - [.NET with Active Directory MFA](#)
 - [Go](#)
 - [Java](#)
 - [Node.js](#)
 - [PHP](#)
 - [Python](#)
 - [Ruby](#)
 - [R](#)
 - [SQL Server drivers](#)

[ADO.NET](#)

[JDBC](#)

[Node.js](#)

[ODBC](#)

[PHP](#)

[Python](#)

[Ruby](#)

[Backup, restore, high availability \(BCDR\)](#)

[Business continuity](#)

[High availability](#)

[Backups](#)

[Automated backups](#)

[Recovery using backups](#)

[Accelerated database recovery](#)

[Long-term backup retention](#)

[Failover groups and geo-replication](#)

[Active geo-replication](#)

[Auto-failover groups](#)

[Configure security for replicas](#)

[Outage recovery guidance](#)

[Recovery drills](#)

[Configure failover group](#)

[Monitor and tune](#)

[Monitor and tune documentation](#)

[Monitoring and performance overview](#)

[Performance monitoring](#)

[Azure SQL Analytics monitoring](#)

[Diagnostic telemetry logging](#)

[Intelligent performance](#)

[Automatic tuning](#)

[Intelligent insights into performance](#)

[Tuning actions and recommendations](#)

Database advisor performance recommendations

Monitor and tune database

- Manual tuning
- Use DMVs to monitor performance
- Use Query store to monitor performance
- Troubleshoot performance with Intelligent Insights
- Use Intelligent Insights diagnostics log
- Monitor In-memory OLTP space

Extended events

- Extended events
- Extended events - event file
- Extended events - ring buffer

Hyperscale performance diagnostics

Resource management in dense elastic pools

Scalability

- Scalability documentation
- Scale Up/Down
- Read Scale-Out
- Database sharding
 - Database sharding
- Glossary
- Elastic client library
- Shard maps
- Query routing
- Manage credentials
- Move sharded data
- Elastic tools FAQ
- Elastic queries
- Elastic transactions

Database features

- Database feature documentation
- Multi-model capabilities

In-memory

[Configure In-Memory OLTP](#)

[Transactional replication](#)

[Temporal tables](#)

[Job automation](#)

How to

[Design a database](#)

[Design database using SSMS](#)

[Design database using .NET](#)

[Load and move data](#)

[Import or export from an Azure VM](#)

[Import a database from a BACPAC file](#)

[Export a database to a BACPAC file](#)

[Copy a database within Azure](#)

[Move resources to a new region](#)

[Load data with BCP](#)

[Load data with ADF](#)

[Develop data applications](#)

[Overview](#)

[Working with JSON data](#)

[Use ASP.NET App Service](#)

[Use Azure Functions](#)

[Use Azure Logic Apps](#)

[Use Spark Connector](#)

[Index with Azure Cognitive Search](#)

[Configure temporal retention policy](#)

[Authenticate app](#)

[Batching for perf](#)

[Connectivity guidance](#)

[DNS aliases](#)

[DNS alias PowerShell](#)

[Ports - ADO.NET](#)

[C and C ++](#)

[Excel](#)

[Server-side CLR/.NET integration](#)

[Java](#)

[Use Spring Data JDBC](#)

[Use Spring Data JPA](#)

[Design data applications](#)

[Design for disaster recovery](#)

[Design for elastic pools](#)

[Design for app upgrades](#)

[Multi-tenant SaaS](#)

[SaaS design patterns](#)

[SaaS video indexer](#)

[SaaS app security](#)

[Multi-tenant SaaS sample application](#)

[Wingtip Tickets sample](#)

[General guidance](#)

[Single application](#)

[Database per tenant](#)

[Disaster recovery using geo-restore](#)

[Disaster recovery using database geo-replication](#)

[Multi-tenant database](#)

[Deploy example app](#)

[Provision tenants](#)

[Monitor database performance](#)

[Run ad-hoc queries](#)

[Manage tenant schema](#)

[ETL for analytics](#)

[Request quota increases](#)

[SQL databases](#)

[SQL databases documentation](#)

[Overview](#)

[Single databases](#)

[Elastic pools](#)

[Database servers](#)

[Quickstarts](#)

[Create](#)

[Using Azure portal](#)

[Using template](#)

[Configure](#)

[Server-level IP firewall rules](#)

[Tutorials](#)

[Migrate to a single/pooled database using DMS](#)

[Add to a failover group](#)

[Implement a geo-distributed database](#)

[Configure active geo-replication](#)

[Set up Azure SQL Data Sync](#)

[Add an elastic pool to a failover group](#)

[Configure security](#)

[Stream data with Stream Analytics](#)

[Migrate a SQLite database to Azure SQL Database Serverless](#)

[Concepts](#)

[Single database resources](#)

[vCore resource limits](#)

[DTU resource limits](#)

[Scale resources](#)

[Elastic pool resources](#)

[vCore resource limits](#)

[DTU resource limits](#)

[Scale resources](#)

[Manage pool resources](#)

[Database server resources](#)

[Database resource limits](#)

[Connectivity architecture](#)

Data sync

[Data Sync Agent](#)

[Replicate schema changes](#)

[Monitor with OMS](#)

[Best practices for Data Sync](#)

[Troubleshoot Data Sync](#)

[Performance monitoring](#)

[Query Performance Insight](#)

[T-SQL differences](#)

[Features](#)

[SQL Data Sync](#)

[Azure Automation](#)

[Management API reference](#)

[Machine Learning Services](#)

[Overview](#)

[Key differences](#)

[Quickstarts](#)

[Create R scripts](#)

[Train and score a model](#)

[Tutorials](#)

[Build a predictive model in R](#)

[1 - Prepare the data](#)

[2 - Create the model](#)

[3 - Deploy the model](#)

[Build a clustering model in R](#)

[1 - Prepare the data](#)

[2 - Create the model](#)

[3 - Deploy the model](#)

[How to](#)

[Write advanced R functions](#)

[Work with R and SQL data](#)

[Add R packages](#)

How to

[Overview](#)

[Migrate to a single/pooled database](#)

[Manage SQL database after migration](#)

[Configure replication](#)

[Configure threat detection](#)

[Configure backup retention using Azure Blob storage](#)

[Configure Elastic Database jobs](#)

[Configure dynamic data masking](#)

[Configure geo-replication - Portal](#)

[Configure security for geo-replicas](#)

[Try in-memory features](#)

[Enable automatic tuning](#)

[Enable e-mail notifications for automatic tuning](#)

[Apply performance recommendations](#)

[Create alerts](#)

[Manage file space](#)

[Use Resource Health for connectivity issues](#)

[Database sharding](#)

[Upgrade elastic database client library](#)

[Create sharded app](#)

[Query horizontally-sharded data](#)

[Multi-shard queries](#)

[Move sharded data](#)

[Security configuration](#)

[Add a shard](#)

[Fix shard map problems](#)

[Migrate sharded database](#)

[Create counters](#)

[Use entity framework](#)

[Use Dapper framework](#)

[Query distributed data](#)

- Query vertically partitioned data
 - Report across scaled-out data tier
 - Query across tables with different schemas
- Elastic jobs
 - Create and manage (PowerShell)
 - Create and manage (T-SQL)
 - Migrate (from old Elastic jobs)
- Samples
 - Azure CLI
 - Azure PowerShell
 - Azure Resource Manager
 - Code samples
- SQL managed instance
 - Overview
 - Managed instance
 - Instance pools
 - FAQ
- Quickstarts
 - Create
 - Using Azure portal
 - Using PowerShell
 - Create instance pools
 - Configure
 - Public endpoint
 - Client VM connection
 - Point-to-site connection
 - Load data
 - Restore database backup
- Tutorials
 - Migrate using DMS
 - Configure security
 - Add managed instance to a failover group

Migrate on-premises users and groups

Transactional replication

MI pub to MI sub

MI pub, MI dist, SQL Sub

Concepts

Management API reference

Resource limits

Connectivity architecture

T-SQL differences

Features

Linked servers

Service Broker

Database mail

How to

Overview

Migrate to Managed instance

Customize time zone

Configure connection types

Migrate TDE cert to Managed instance

Restore to a point in time

Determine size of managed instance subnet

Create new VNet and subnet for managed instance

Configure existing VNet and subnet for managed instance

Delete subnet after deleting managed instance

Configure custom DNS

Sync network configuration

Find management endpoint IP address

Verify built-in firewall protection

Configure threat detection

Connect applications

Samples

Azure CLI

[Azure PowerShell](#)

[Azure Resource Manager](#)

[Code samples](#)

[Reference](#)

[Azure CLI](#)

[Azure PowerShell](#)

[.NET](#)

[Java](#)

[T-SQL language reference](#)

[REST](#)

[Resource Manager templates for SQL](#)

[SQL Server tools](#)

[SQL Server Management Studio \(SSMS\)](#)

[SQL Server Data Tools \(SSDT\)](#)

[BCP](#)

[SQLCMD](#)

[SqlPackage](#)

[SQL Database Management Library package](#)

[Resources](#)

[Glossary of terms](#)

[Build your skills with Microsoft Learn](#)

[SQL Server Blog](#)

[Microsoft Azure Blog](#)

[Azure Roadmap](#)

[Public data sets](#)

[Pricing](#)

[MSDN forum](#)

[Stack Overflow](#)

[Troubleshoot](#)

[Connectivity issues](#)

[Problem resolution](#)

[Import/Export service hangs](#)

[Videos](#)

[Service updates](#)

[Architecture center](#)

[Customer stories](#)

What is the Azure SQL Database service?

11/7/2019 • 20 minutes to read • [Edit Online](#)

Azure SQL Database is a general-purpose relational database, provided as a managed service. With it, you can create a highly available and high-performance data storage layer for the applications and solutions in Azure. SQL Database can be the right choice for a variety of modern cloud applications because it enables you to process both relational data and [non-relational structures](#), such as graphs, JSON, spatial, and XML.

It's based on the latest stable version of the [Microsoft SQL Server database engine](#). You can use advanced query processing features, such as [high-performance in-memory technologies](#) and [intelligent query processing](#). In fact, the newest capabilities of SQL Server are released first to SQL Database, and then to SQL Server itself. You get the newest SQL Server capabilities with no overhead for patching or upgrading, tested across millions of databases.

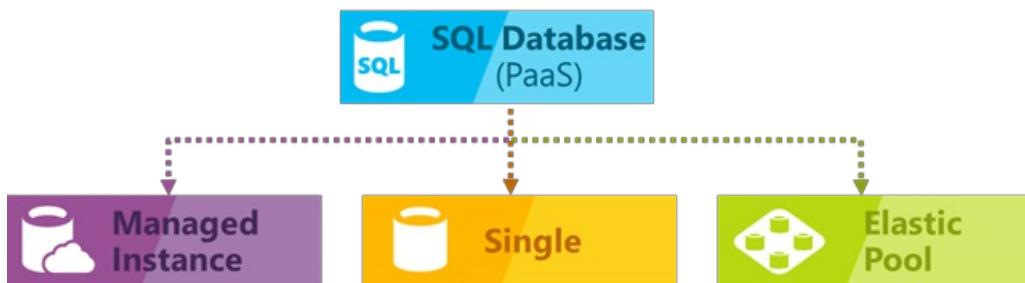
SQL Database enables you to easily define and scale performance within two different purchasing models: a [vCore-based purchasing model](#) and a [DTU-based purchasing model](#). SQL Database is a fully managed service that has built-in high availability, backups, and other common maintenance operations. Microsoft handles all patching and updating of the SQL and operating system code. You don't have to manage the underlying infrastructure.

NOTE

For relevant terms and their definitions, see the [SQL Database terms glossary](#).

Deployment models

Azure SQL Database provides the following deployment options for an Azure SQL database:



- [Single database](#) represents a fully managed, isolated database. You might use this option if you have modern cloud applications and microservices that need a single reliable data source. A single database is similar to a [contained database](#) in [Microsoft SQL Server Database Engine](#).
- [Managed instance](#) is a fully managed instance of the [Microsoft SQL Server Database Engine](#). It contains a set of databases that can be used together. Use this option for easy migration of on-premises SQL Server databases to the Azure cloud, and for applications that need to use the database features that SQL Server Database Engine provides.
- [Elastic pool](#) is a collection of [single databases](#) with a shared set of resources, such as CPU or memory. Single databases can be moved into and out of an elastic pool.

IMPORTANT

To understand the feature differences between SQL Database and SQL Server, as well as the differences among different Azure SQL Database deployment options, see [SQL Database features](#).

SQL Database delivers predictable performance with multiple resource types, service tiers, and compute sizes. It provides dynamic scalability with no downtime, built-in intelligent optimization, global scalability and availability, and advanced security options. These capabilities allow you to focus on rapid app development and accelerating your time-to-market, rather than on managing virtual machines and infrastructure. The SQL Database service is currently in 38 datacenters around the world, so you can run your database in a datacenter near you.

Scalable performance and pools

You can define the amount of resources assigned.

- With single databases, each database is isolated from others and is portable. Each has its own guaranteed amount of compute, memory, and storage resources. The amount of the resources assigned to the database is dedicated to that database, and isn't shared with other databases in Azure. You can dynamically [scale single database resources](#) up and down. The single database option provides different compute, memory, and storage resources for different needs. For example, you can get 1 to 80 vCores, or 32 GB to 4 TB. The [hyperscale service tier](#) for single database enables you to scale to 100 TB, with fast backup and restore capabilities.
- With elastic pools, you can assign resources that are shared by all databases in the pool. You can create a new database, or move the existing single databases into a resource pool to maximize the use of resources and save money. This option also gives you the ability to dynamically [scale elastic pool resources](#) up and down.
- With managed instances, each managed instance is isolated from other instances with guaranteed resources. Within a managed instance, the instance databases share a set of resources. You can dynamically [scale managed instance resources](#) up and down.

You can build your first app on a small, single database at a low cost per month in the general-purpose service tier. You can then change its service tier manually or programmatically at any time to the business-critical service tier, to meet the needs of your solution. You can adjust performance without downtime to your app or to your customers. Dynamic scalability enables your database to transparently respond to rapidly changing resource requirements. You pay for only the resources that you need when you need them.

Dynamic scalability is different from *autoscale*. Autoscale is when a service scales automatically based on criteria, whereas dynamic scalability allows for manual scaling without downtime. The single database option supports manual dynamic scalability, but not autoscale. For a more automatic experience, consider using elastic pools, which allow databases to share resources in a pool based on individual database needs. Another option is to use scripts that can help automate scalability for a single database. For an example, see [Use PowerShell to monitor and scale a single database](#).

Purchasing models

SQL Database offers the following purchasing models:

- The [vCore-based purchasing model](#) lets you choose the number of vCores, the amount of memory, and the amount and speed of storage. The vCore-based purchasing model also allows you to use [Azure Hybrid Benefit for SQL Server](#) to gain cost savings. For more information about the Azure Hybrid Benefit, see the "Frequently asked questions" section later in this article.
- The [DTU-based purchasing model](#) offers a blend of compute, memory, and I/O resources in three service tiers, to support light to heavy database workloads. Compute sizes within each tier provide a different mix of these resources, to which you can add additional storage resources.
- The [serverless model](#) automatically scales compute based on workload demand, and bills for the amount of

compute used per second. The serverless compute tier also automatically pauses databases during inactive periods when only storage is billed, and automatically resumes databases when activity returns.

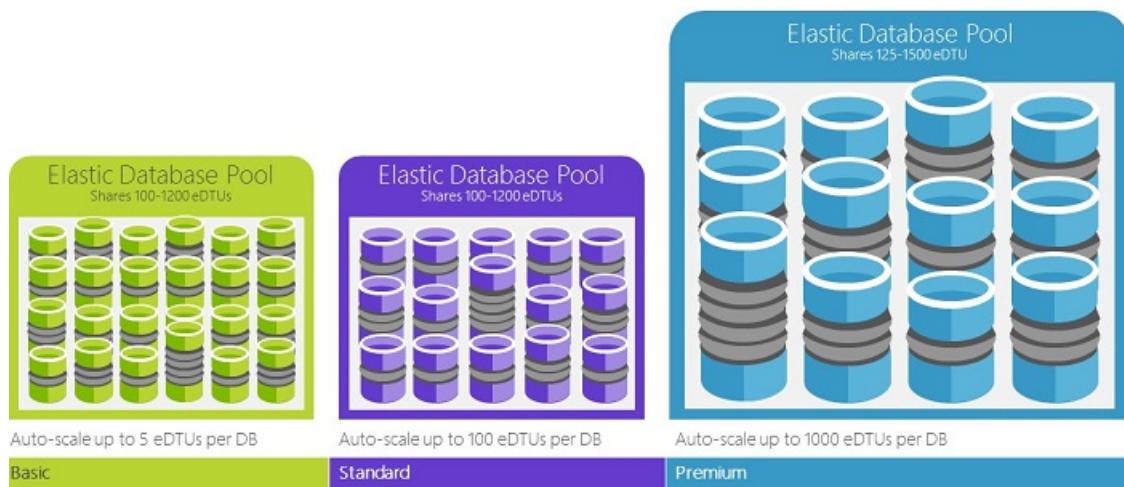
Service tiers

Azure SQL Database offers three service tiers that are designed for different types of applications:

- [General Purpose/Standard](#) service tier designed for common workloads. It offers budget-oriented balanced compute and storage options.
- [Business Critical/Premium](#) service tier designed for OLTP applications with high transaction rate and lowest-latency I/O. It offers the highest resilience to failures by using several isolated replicas.
- [Hyperscale](#) service tier designed for very large OLTP database and the ability to auto-scale storage and scale compute fluidly.

Elastic pools to maximize resource utilization

For many businesses and applications, being able to create single databases and dial performance up or down on demand is enough, especially if usage patterns are relatively predictable. Unpredictable usage patterns can make it hard to manage costs and your business model. [Elastic pools](#) are designed to solve this problem. You allocate performance resources to a pool rather than an individual database. You pay for the collective performance resources of the pool rather than for single database performance.



With elastic pools, you don't need to focus on dialing database performance up and down as demand for resources fluctuates. The pooled databases consume the performance resources of the elastic pool as needed. Pooled databases consume but don't exceed the limits of the pool, so your cost remains predictable even if individual database usage doesn't.

You can [add and remove databases to the pool](#), scaling your app from a handful of databases to thousands, all within a budget that you control. You can also control the minimum and maximum resources available to databases in the pool, to ensure that no database in the pool uses all the pool resources, and that every pooled database has a guaranteed minimum amount of resources. To learn more about design patterns for software as a service (SaaS) applications that use elastic pools, see [Design patterns for multi-tenant SaaS applications with SQL Database](#).

Scripts can help with monitoring and scaling elastic pools. For an example, see [Use PowerShell to monitor and scale a SQL elastic pool in Azure SQL Database](#).

IMPORTANT

A managed instance doesn't support elastic pools. Rather, a managed instance is a collection of instance databases that share managed instance resources.

Blend single databases with pooled databases

You can blend single databases with elastic pools, and change the service tiers of single databases and elastic pools to adapt to your situation. You can also mix and match other Azure services with SQL Database to meet your unique app design needs, drive cost and resource efficiencies, and unlock new business opportunities.

Extensive monitoring and alerting capabilities

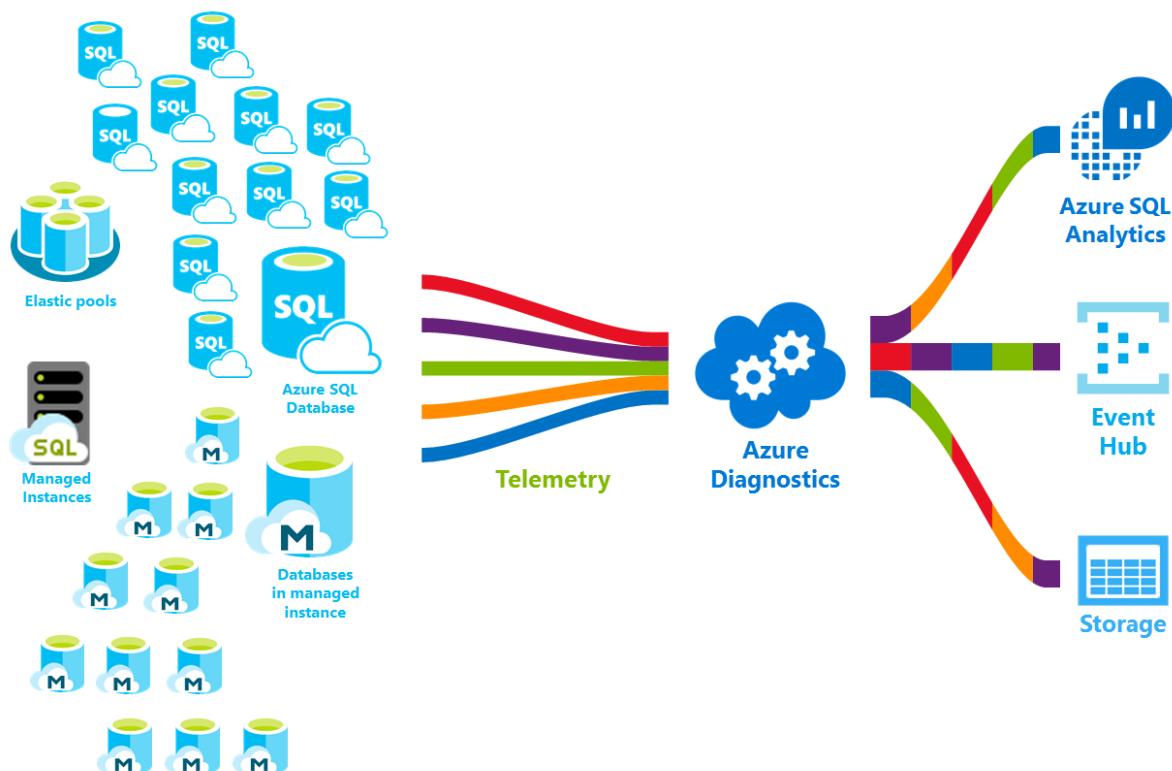
Azure SQL Database provides advanced monitoring and troubleshooting features that help you get deeper insights into workload characteristics. These features and tools include:

- The built-in monitoring capabilities provided by the latest version of SQL Server Database Engine. They enable you to find real-time performance insights.
- PaaS monitoring capabilities provided by Azure that enable you to monitor and troubleshoot a large number of database instances.

[Query Store](#), a built-in SQL Server monitoring feature, records the performance of your queries in real time, and enables you to identify the potential performance issues and the top resource consumers. Automatic tuning and recommendations provide advice regarding the queries with the regressed performance and missing or duplicated indexes. Automatic tuning in SQL Database enables you to either manually apply the scripts that can fix the issues, or let SQL Database apply the fix. SQL Database can also test and verify that the fix provides some benefit, and retain or revert the change depending on the outcome. In addition to Query Store and automatic tuning capabilities, you can use standard [DMVs and XEvent](#) to monitor the workload performance.

Azure provides [built-in performance monitoring](#) and [alerting](#) tools, combined with performance ratings, that enable you to monitor the status of thousands of databases. Using these tools, you can quickly assess the impact of scaling up or down, based on your current or projected performance needs. Additionally, SQL Database can [emit metrics and diagnostic logs](#) for easier monitoring. You can configure SQL Database to store resource usage, workers and sessions, and connectivity into one of these Azure resources:

- **Azure Storage:** For archiving vast amounts of telemetry for a small price.
- **Azure Event Hubs:** For integrating SQL Database telemetry with your custom monitoring solution or hot pipelines.
- **Azure Monitor logs:** For a built-in monitoring solution with reporting, alerting, and mitigating capabilities.



Availability capabilities

In a traditional SQL Server environment, you generally have at least two machines locally set up. These machines have exact, synchronously maintained, copies of the data to protect against a failure of a single machine or component. This environment provides high availability, but it doesn't protect against a natural disaster destroying your datacenter.

Disaster recovery assumes that a catastrophic event is geographically localized enough to have another machine or set of machines with a copy of your data far away. In SQL Server, you can use Always On Availability Groups running in async mode to get this capability. People often don't want to wait for replication to happen that far away before committing a transaction, so there's potential for data loss when you do unplanned failovers.

Databases in the premium and business critical service tiers already [do something very similar](#) to the synchronization of an availability group. Databases in lower service tiers provide redundancy through storage by using a [different but equivalent mechanism](#). Built-in logic helps protect against a single machine failure. The active geo-replication feature gives you the ability to protect against disaster where a whole region is destroyed.

Azure Availability Zones tries to protect against the outage of a single datacenter building within a single region. It helps you protect against the loss of power or network to a building. In SQL Database, you place the different replicas in different availability zones (different buildings, effectively).

In fact, the service level agreement ([SLA](#)) of Azure, powered by a global network of Microsoft-managed datacenters, helps keep your app running 24/7. The Azure platform fully manages every database, and it guarantees no data loss and a high percentage of data availability. Azure automatically handles patching, backups, replication, failure detection, underlying potential hardware, software or network failures, deploying bug fixes, failovers, database upgrades, and other maintenance tasks. Standard availability is achieved by a separation of compute and storage layers. Premium availability is achieved by integrating compute and storage on a single node for performance, and then implementing technology similar to Always On Availability Groups. For a full discussion of the high availability capabilities of Azure SQL Database, see [SQL Database availability](#).

In addition, SQL Database provides built-in [business continuity and global scalability](#) features. These include:

- [Automatic backups](#):

SQL Database automatically performs full, differential, and transaction log backups of SQL databases to enable you to restore to any point in time. For single databases and pooled databases, you can configure SQL Database to store full database backups to Azure Storage for long-term backup retention. For managed instances, you can also perform copy-only backups for long-term backup retention.

- [Point-in-time restores](#):

All SQL Database deployment options support recovery to any point in time within the automatic backup retention period for any SQL database.

- [Active geo-replication](#):

The single database and pooled databases options allow you to configure up to four readable secondary databases in either the same or globally distributed Azure datacenters. For example, if you have a SaaS application with a catalog database that has a high volume of concurrent read-only transactions, use active geo-replication to enable global read scale. This removes bottlenecks on the primary that are due to read workloads. For managed instances, use auto-failover groups.

- [Auto-failover groups](#):

All SQL Database deployment options allow you to use failover groups to enable high availability and load balancing at global scale. This includes transparent geo-replication and failover of large sets of databases, elastic pools, and managed instances. Failover groups enable the creation of globally distributed SaaS applications, with minimal administration overhead. This leaves all the complex

monitoring, routing, and failover orchestration to SQL Database.

- [Zone-redundant databases](#):

SQL Database allows you to provision premium or business critical databases or elastic pools across multiple availability zones. Because these databases and elastic pools have multiple redundant replicas for high availability, placing these replicas into multiple availability zones provides higher resilience. This includes the ability to recover automatically from the datacenter scale failures, without data loss.

Built-in intelligence

With SQL Database, you get built-in intelligence that helps you dramatically reduce the costs of running and managing databases, and that maximizes both performance and security of your application. Running millions of customer workloads around the clock, SQL Database collects and processes a massive amount of telemetry data, while also fully respecting customer privacy. Various algorithms continuously evaluate the telemetry data so that the service can learn and adapt with your application.

Automatic performance monitoring and tuning

SQL Database provides detailed insight into the queries that you need to monitor. SQL Database learns about your database patterns, and enables you to adapt your database schema to your workload. SQL Database provides [performance tuning recommendations](#), where you can review tuning actions and apply them.

However, constantly monitoring a database is a hard and tedious task, especially when dealing with many databases. [Intelligent Insights](#) does this job for you by automatically monitoring SQL Database performance at scale. It informs you of performance degradation issues, it identifies the root cause of each issue, and provides performance improvement recommendations when possible.

Managing a huge number of databases might be impossible to do efficiently even with all available tools and reports that SQL Database and Azure provide. Instead of monitoring and tuning your database manually, you might consider delegating some of the monitoring and tuning actions to SQL Database by using [automatic tuning](#). SQL Database automatically applies recommendations, tests, and verifies each of its tuning actions to ensure the performance keeps improving. This way, SQL Database automatically adapts to your workload in a controlled and safe way. Automatic tuning means that the performance of your database is carefully monitored and compared before and after every tuning action. If the performance doesn't improve, the tuning action is reverted.

Many of our partners that run [SaaS multi-tenant apps](#) on top of SQL Database are relying on automatic performance tuning to make sure their applications always have stable and predictable performance. For them, this feature tremendously reduces the risk of having a performance incident in the middle of the night. In addition, because part of their customer base also uses SQL Server, they're using the same indexing recommendations provided by SQL Database to help their SQL Server customers.

Two automatic tuning aspects are [available in SQL Database](#):

- **Automatic index management:** Identifies indexes that should be added in your database, and indexes that should be removed.
- **Automatic plan correction:** Identifies problematic plans and fixes SQL plan performance problems.

Adaptive query processing

You can use [adaptive query processing](#), including interleaved execution for multi-statement table-valued functions, batch mode memory grant feedback, and batch mode adaptive joins. Each of these adaptive query processing features applies similar "learn and adapt" techniques, helping further address performance issues related to historically intractable query optimization problems.

Advanced security and compliance

SQL Database provides a range of [built-in security and compliance features](#) to help your application meet various security and compliance requirements.

IMPORTANT

Microsoft has certified Azure SQL Database (all deployment options) against a number of compliance standards. For more information, see the [Microsoft Azure Trust Center](#), where you can find the most current list of SQL Database compliance certifications.

Advance threat protection

Advanced data security is a unified package for advanced SQL security capabilities. It includes functionality for discovering and classifying sensitive data, managing your database vulnerabilities, and detecting anomalous activities that might indicate a threat to your database. It provides a single location for enabling and managing these capabilities.

- [Data discovery and classification](#):

This feature provides capabilities built into Azure SQL Database for discovering, classifying, labeling, and protecting the sensitive data in your databases. It provides visibility into your database classification state, and tracks the access to sensitive data within the database and beyond its borders.

- [Vulnerability assessment](#):

This service can discover, track, and help you remediate potential database vulnerabilities. It provides visibility into your security state, and includes actionable steps to resolve security issues, and enhance your database fortifications.

- [Threat detection](#):

This feature detects anomalous activities that indicate unusual and potentially harmful attempts to access or exploit your database. It continuously monitors your database for suspicious activities, and provides immediate security alerts on potential vulnerabilities, SQL injection attacks, and anomalous database access patterns. Threat detection alerts provide details of the suspicious activity, and recommend action on how to investigate and mitigate the threat.

Auditing for compliance and security

[Auditing](#) tracks database events and writes them to an audit log in your Azure storage account. Auditing can help you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that might indicate business concerns or suspected security violations.

Data encryption

SQL Database helps secure your data by providing encryption. For data in motion, it uses [transport layer security](#). For data at rest, it uses [transparent data encryption](#). For data in use, it uses [always encrypted](#).

Azure Active Directory integration and multi-factor authentication

SQL Database enables you to centrally manage identities of database user and other Microsoft services with [Azure Active Directory integration](#). This capability simplifies permission management and enhances security. Azure Active Directory supports [multi-factor authentication](#) to increase data and application security, while supporting a single sign-in process.

Easy-to-use tools

SQL Database makes building and maintaining applications easier and more productive. SQL Database allows you to focus on what you do best: building great apps. You can manage and develop in SQL Database by using tools and skills you already have.

- [The Azure portal](#):

A web-based application for managing all Azure services.

- [SQL Server Management Studio](#):

A free, downloadable client application for managing any SQL infrastructure, from SQL Server to SQL Database.

- [SQL Server Data Tools in Visual Studio](#):

A free, downloadable client application for developing SQL Server relational databases, SQL databases, Integration Services packages, Analysis Services data models, and Reporting Services reports.

- [Visual Studio Code](#):

A free, downloadable, open-source code editor for Windows, macOS, and Linux. It supports extensions, including the [mssql extension](#) for querying Microsoft SQL Server, Azure SQL Database, and Azure SQL Data Warehouse.

SQL Database supports building applications with Python, Java, Node.js, PHP, Ruby, and .NET on macOS, Linux, and Windows. SQL Database supports the same [connection libraries](#) as SQL Server.

Create and manage Azure SQL resources with the Azure portal

The Azure portal provides a single page where you can manage [all of your Azure SQL resources](#) including your SQL virtual machines.

To access the **Azure SQL resources** page, from the Azure portal menu, select **Azure SQL** or search for and select **Azure SQL** in any page.

NOTE

Azure SQL provides a quick and easy way to access all of your SQL databases, elastic pools, database servers, SQL managed instances, and SQL virtual machines. Azure SQL is not a service or resource.

To manage existing resources, select the desired item in the list. To create new Azure SQL resources, select **+ Add**.

Azure SQL - Microsoft Azure

All services > Azure SQL

Azure SQL

+ Add Reservations Edit columns Refresh Export to CSV Assign tags Delete Feedback

Filter by name... Subscription == all Resource group == all Location == all Add filter

No grouping

Name	Resource type	Service tier	Resource group	Location
contoso-database (contoso-sq...)	SQL database	Basic: 5 DTUs	contososql	East US
contoso-sql-server	SQL server	--	contososql	East US

< Previous Page 1 of 1 Next >

After selecting **+ Add**, view additional information about the different options by selecting **Show details** on any tile.

All services > Azure SQL > Select SQL deployment option

Select SQL deployment option

Feedback

How do you plan to use the service?

SQL databases
Best for modern cloud applications. Hyperscale and serverless options are available.

Resource type: Single database

Create Show details

SQL managed instances
Best for most migrations to the cloud. Lift-and-shift ready.

Resource type: Single instance

Create Show details

SQL virtual machines
Best for migrations and applications requiring OS-level access. Lift-and-shift ready.

Image: Please select an offer

Create Show details

For details, see:

- [Create a single database](#)
- [Create an elastic pool](#)
- [Create a managed instance](#)

- [Create a SQL virtual machine](#)

SQL Database frequently asked questions

What is the current version of SQL Database?

The current version of SQL Database is V12. Version V11 has been retired.

Can I control when patching downtime occurs?

No. The impact of patching is generally not noticeable if you [employ retry logic](#) in your app. For more information, see [Planning for Azure maintenance events in Azure SQL Database](#).

Azure Hybrid Benefit questions

Are there dual-use rights with Azure Hybrid Benefit for SQL Server?

You have 180 days of dual use rights of the license to ensure migrations are running seamlessly. After that 180-day period, you can only use the SQL Server license in the cloud in SQL Database. You no longer have dual use rights on-premises and in the cloud.

How does Azure Hybrid Benefit for SQL Server differ from license mobility?

We offer license mobility benefits to SQL Server customers with Software Assurance. This allows reassignment of their licenses to a partner's shared servers. You can use this benefit on Azure IaaS and AWS EC2.

Azure Hybrid Benefit for SQL Server differs from license mobility in two key areas:

- It provides economic benefits for moving highly virtualized workloads to Azure. SQL Server Enterprise Edition customers can get four cores in Azure in the General Purpose SKU for every core they own on-premises for highly virtualized applications. License mobility doesn't allow any special cost benefits for moving virtualized workloads to the cloud.
- It provides for a PaaS destination on Azure (SQL Database managed instance) that's highly compatible with SQL Server on-premises.

What are the specific rights of the Azure Hybrid Benefit for SQL Server?

SQL Database customers have the following rights associated with Azure Hybrid Benefit for SQL Server:

LICENSE FOOTPRINT	WHAT DOES AZURE HYBRID BENEFIT FOR SQL SERVER GET YOU?
SQL Server Enterprise Edition core customers with SA	<ul style="list-style-type: none"> • Can pay base rate on either General Purpose or Business Critical SKU • 1 core on-premises = 4 cores in General Purpose SKU • 1 core on-premises = 1 core in Business Critical SKU
SQL Server Standard Edition core customers with SA	<ul style="list-style-type: none"> • Can pay base rate on General Purpose SKU only • 1 core on-premises = 1 core in General Purpose SKU

Engage with the SQL Server engineering team

- [DBA Stack Exchange](#): Ask database administration questions.
- [Stack Overflow](#): Ask development questions.
- [MSDN Forums](#): Ask technical questions.
- [Feedback](#): Report bugs and request feature.
- [Reddit](#): Discuss SQL Server.

Next steps

- See the [pricing page](#) for cost comparisons and calculators regarding single databases and elastic pools.
- See these quickstarts to get started:
 - [Create a SQL database in the Azure portal](#)
 - [Create a SQL database with the Azure CLI](#)
 - [Create a SQL database using PowerShell](#)
- For a set of Azure CLI and PowerShell samples, see:
 - [Azure CLI samples for SQL Database](#)
 - [Azure PowerShell samples for SQL Database](#)
- For information about new capabilities as they're announced, see [Azure Roadmap for SQL Database](#).
- See the [Azure SQL Database blog](#), where SQL Server product team members blog about SQL Database news and features.

Choose the right deployment option in Azure SQL

11/7/2019 • 16 minutes to read • [Edit Online](#)

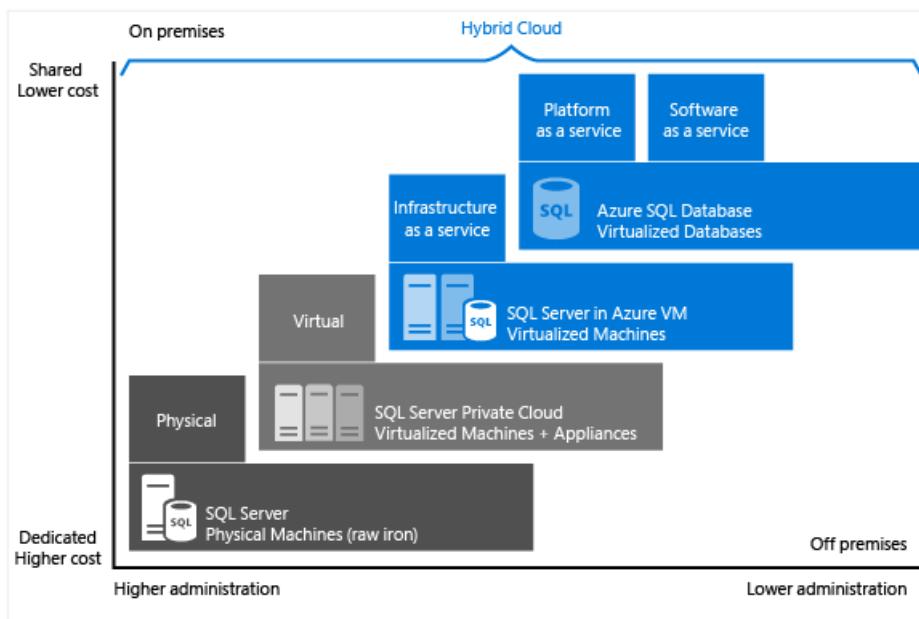
Learn how each deployment option fits into Microsoft's Azure SQL data platform, and get help matching the right option for your business requirements. Whether you prioritize cost savings or minimal administration, this article can help you decide which approach delivers against the business requirements you care about most.

Microsoft's Azure SQL data platform

From lift-and-shift migrations, to modernization of existing applications, to building modern cloud services, Azure SQL is a modern SQL platform that offers several deployment options, powered by an industry leading Microsoft SQL Server engine. Azure SQL is designed to support a wide variety of application patterns with different levels of control over the underlying platform to meet the most demanding migration and modernization requirements.

Azure SQL eliminates the complexity of managing diverse collections of SQL Server-based applications at scale by providing a single, unified management experience.

One of the first things to understand in any discussion of Azure versus on-premises SQL Server databases is that you can use it all. Microsoft's data platform leverages SQL Server technology and makes it available across physical on-premises machines, private cloud environments, third-party hosted private cloud environments, and the public cloud. SQL Server on Azure virtual machines (SQL virtual machines) enables you to meet unique and diverse business needs through a combination of on-premises and cloud-hosted deployments, while using the same set of server products, development tools, and expertise across these environments.



As seen in the diagram, each offering can be characterized by the level of administration you have over the infrastructure, and by the degree of cost efficiency.

In Azure, you can have your SQL Server workloads running as a hosted service ([PaaS](#)), or a hosted infrastructure ([IaaS](#)). Within PaaS, you have multiple deployment options and service tiers within each deployment option. The key question that you need to ask when deciding between PaaS or IaaS is do you want to manage your database, apply patches, and take backups, or do you want to delegate these operations to Azure?

Depending on the answer, you have the following options:

- **SQL databases:** Best for modern cloud applications that want to use the latest stable SQL Server features and have time constraints in development and marketing. A fully-managed SQL database engine, based on

the latest stable Enterprise Edition of SQL Server. This is a relational database-as-a-service (DBaaS) hosted in the Azure cloud that falls into the industry category of *Platform-as-a-Service (PaaS)*. SQL database has multiple deployment options, each of which is built on standardized hardware and software that is owned, hosted, and maintained by Microsoft. With SQL Server, you can use built-in features and functionality that require extensive configuration (either on-premises or in an Azure virtual machine). When using SQL Database, you pay-as-you-go with options to scale up or out for greater power with no interruption. SQL Database has some additional features that are not available in SQL Server, such as built-in high availability, intelligence, and management.

Databases offer the following deployment options:

- As a **single database** with its own set of resources managed via a database server. A single database is similar to a **contained database** in SQL Server. This option is optimized for modern application development of new cloud-born applications. [Hyperscale](#) and [serverless](#) options are available.
- An **elastic pool**, which is a collection of databases with a shared set of resources managed via a database server. Single databases can be moved into and out of an elastic pool. This option is optimized for modern application development of new cloud-born applications using the multi-tenant SaaS application pattern. Elastic pools provide a cost-effective solution for managing the performance of multiple databases that have variable usage patterns.
- A **database server**, which is used to manage groups of single databases and elastic pools. Database servers act as a central administrative point for multiple single or pooled databases, [logins](#), [firewall rules](#), [auditing rules](#), [threat detection policies](#), and [failover groups](#).
- **SQL managed instances:** Best for most migrations to the cloud. Managed instance is a collection of system and user databases with a shared set of resources that is lift-and-shift ready. Best for new applications or existing on-premises applications that want to use the latest stable SQL Server features and that are migrated to the cloud with minimal changes. A managed instance is similar to an instance of the [Microsoft SQL Server database engine](#) offering shared resources for databases and additional instance-scoped features. Managed instance supports database migration from on-premises with minimal to no database change. This option provides all of the PaaS benefits of Azure SQL Database but adds capabilities that were previously only available in SQL VMs. This includes a native virtual network (VNet) and near 100% compatibility with on-premises SQL Server. Managed instances provide full SQL Server access and feature compatibility for migrating SQL Servers to Azure.
- **SQL virtual machines:** Best for migrations and applications requiring OS-level access. SQL virtual machines are lift-and-shift ready for existing applications that require fast migration to the cloud with minimal changes or no changes. SQL virtual machines offer full administrative control over the SQL Server instance and underlying OS for migration to Azure. Rapid development and test scenarios when you do not want to buy on-premises non-production SQL Server hardware. SQL virtual machines fall into the industry category *Infrastructure-as-a-Service (IaaS)* and allows you to run SQL Server inside a fully-managed virtual machine (VM) in the Azure cloud. SQL virtual machines also run on standardized hardware that is owned, hosted, and maintained by Microsoft. When using SQL virtual machines, you can either pay-as-you-go for a SQL Server license already included in a SQL Server image or easily use an existing license. You can also stop or resume the VM as needed. SQL Server installed and hosted in the cloud runs on Windows Server or Linux virtual machines running on Azure, also known as an infrastructure as a service (IaaS). SQL virtual machines is a good option for migrating on-premises SQL Server databases and applications without any database change. All recent versions and editions of SQL Server are available for installation in an IaaS virtual machine. The most significant difference from SQL databases and SQL managed instances, is that SQL Server VMs allow full control over the database engine. You can choose when to start maintenance/patching, change the recovery model to simple or bulk-logged, pause or start the service when needed, and you can fully customize the SQL Server database engine. With this additional control comes the added responsibility to manage the virtual machine.

Optimized for migrating existing applications to Azure or extending existing on-premises applications to the

cloud in hybrid deployments. In addition, you can use SQL Server in a virtual machine to develop and test traditional SQL Server applications. With SQL virtual machines, you have the full administrative rights over a dedicated SQL Server instance and a cloud-based VM. It is a perfect choice when an organization already has IT resources available to maintain the virtual machines. These capabilities allow you to build a highly customized system to address your application's specific performance and availability requirements.

Additional differences are listed in the following table, but ***both databases and managed instances are optimized to reduce overall management costs to the minimum for provisioning and managing many databases.*** It reduces ongoing administration costs because you do not have to manage any virtual machines, operating system or database software. You do not have to manage upgrades, high availability, or backups. In general, Azure SQL Database can dramatically increase the number of databases managed by a single IT or development resource. [Elastic pools](#) also support SaaS multi-tenant application architectures with features including tenant isolation and the ability to scale to reduce costs by sharing resources across databases. [Managed instance](#) provides support for instance-scoped features enabling easy migration of existing applications, as well as sharing resources amongst databases.

SQL DATABASES	SQL MANAGED INSTANCES	SQL VIRTUAL MACHINES
Supports most on-premises database-level capabilities. The most commonly used SQL Server features are available. 99.995% availability guaranteed. Built-in backups, patching, recovery. Latest stable Database Engine version. Ability to assign necessary resources (CPU/storage) to individual databases. Built-in advanced intelligence and security. Online change of resources (CPU/storage).	Supports almost all on-premises instance-level and database-level capabilities. High compatibility with SQL Server on-premises. 99.99% availability guaranteed. Built-in backups, patching, recovery. Latest stable Database Engine version. Easy migration from SQL Server. Private IP address within Azure VNet. Built-in advanced intelligence and security. Online change of resources (CPU/storage).	You have full control over the SQL Server engine. Supports all on-premises capabilities. Up to 99.99% availability. Full parity with the matching version of on-premises SQL Server. Fixed, well-known database engine version. Easy migration from SQL Server on-premises. Private IP address within Azure VNet. You have ability to deploy application or services on the host where SQL Server is placed.
Migration from SQL Server might be hard. Some SQL Server features are not available. No guaranteed exact maintenance time (but nearly transparent). Compatibility with the SQL Server version can be achieved only using database compatibility levels. Private IP address cannot be assigned (you can limit the access using firewall rules).	There is still some minimal number of SQL Server features that are not available. No guaranteed exact maintenance time (but nearly transparent). Compatibility with the SQL Server version can be achieved only using database compatibility levels.	You need to manage your backups and patches. You need to implement your own High-Availability solution. There is a downtime while changing the resources(CPU/storage)
Databases of up to 100 TB.	Up to 8 TB.	SQL Server instances with up to 256 TB of storage. The instance can support as many databases as needed.

SQL DATABASES	SQL MANAGED INSTANCES	SQL VIRTUAL MACHINES
On-premises application can access data in Azure SQL Database.	Native virtual network implementation and connectivity to your on-premises environment using Azure Express Route or VPN Gateway.	With SQL virtual machines, you can have applications that run partly in the cloud and partly on-premises. For example, you can extend your on-premises network and Active Directory Domain to the cloud via Azure Virtual Network . For more information on hybrid cloud solutions, see Extending on-premises data solutions to the cloud .

Business motivations for choosing databases, managed instances, or SQL virtual machines

There are several factors that can influence your decision to choose between the different data offerings:

- **Cost** - Both PaaS and IaaS option include base price that cover underlying infrastructure and licensing. However, with IaaS option you need to invest additional time and resources to manage your database, while in PaaS you are getting these administration features included in the price. IaaS option enables you to shut down your resources while you are not using them to decrease the cost, while PaaS version is always running unless if you drop and re-create your resources when they are needed.
- **Administration** - PaaS options reduce the amount of time that you need to invest to administer the database. However, it also limits the range of custom administration tasks and scripts that you can perform or run. For example, the CLR is not supported with single or pooled databases, but is supported for a managed instance. Also, no deployment options in PaaS support the use of trace flags.
- **Service-Level Agreement** - Both IaaS and PaaS provide high, industry standard SLA. PaaS option guarantees 99.99% SLA, while IaaS guarantees 99.95% SLA for infrastructure, meaning that you need to implement additional mechanisms to ensure availability of your databases. You can implement High-availability solution at 99.99% by creating an additional SQL Server in VM and configure AlwaysOn Availability groups.
- **Time to move to Azure** - SQL Server in Azure VM is the exact match of your environment, so migration from on-premises to Azure SQL VM is not different than moving the databases from one on-premises server to another. Managed instance also enables extremely easy migration; however, there might be some changes that you need to apply before you migrate to a managed instance.

These factors will be discussed in more details in the following sections.

Cost

Whether you're a startup that is strapped for cash, or a team in an established company that operates under tight budget constraints, limited funding is often the primary driver when deciding how to host your databases. In this section, you learn about the billing and licensing basics in Azure with regards to these two relational database options: SQL Database and SQL virtual machines. You also learn about calculating the total application cost.

Billing and licensing basics

Currently, **SQL Database** is sold as a service and is available with several deployment options and in several service tiers with different prices for resources, all of which are billed hourly at a fixed rate based on the service tier and compute size you choose. For the latest information on the current supported service tiers, compute sizes, and storage amounts, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#).

- With SQL database, you can choose a service tier that fits your needs from a wide range of prices starting from 5\$/month for basic tier.
- You can create [elastic pools](#) to share resources among database instances to reduce costs and accommodate usage spikes.

- With SQL managed instance, you can also bring your own license. For more information on bring-your-own licensing, see [License Mobility through Software Assurance on Azure](#) or use [Azure Hybrid Benefit calculator](#) to see how to **save up to 40%**.

In addition, you are billed for outgoing Internet traffic at regular [data transfer rates](#). You can dynamically adjust service tiers and compute sizes to match your application's varied throughput needs.

With **SQL databases and SQL managed instances**, the database software is automatically configured, patched, and upgraded by Microsoft, which reduces your administration costs. In addition, its [built-in backup](#) capabilities help you achieve significant cost savings, especially when you have a large number of databases.

With **SQL virtual machines**, you can use any of the platform-provided SQL Server images (which includes a license) or bring your SQL Server license. All the supported SQL Server versions (2008R2, 2012, 2014, 2016) and editions (Developer, Express, Web, Standard, Enterprise) are available. In addition, Bring-Your-Own-License versions (BYOL) of the images are available. When using the Azure provided images, the operational cost depends on the VM size and the edition of SQL Server you choose. Regardless of VM size or SQL Server edition, you pay per-minute licensing cost of SQL Server and the Windows or Linux Server, along with the Azure Storage cost for the VM disks. The per-minute billing option allows you to use SQL Server for as long as you need without buying additional SQL Server licenses. If you bring your own SQL Server license to Azure, you are charged for server and storage costs only. For more information on bring-your-own licensing, see [License Mobility through Software Assurance on Azure](#). In addition, you are billed for outgoing Internet traffic at regular [data transfer rates](#).

Calculating the total application cost

When you start using a cloud platform, the cost of running your application includes the cost for new development and ongoing administration costs, plus the public cloud platform service costs.

For more information on pricing, see the following resources:

- [SQL Database pricing](#)
- [Virtual machine pricing](#) for [SQL](#) and for [Windows](#)
- [Azure Pricing Calculator](#)

Administration

For many businesses, the decision to transition to a cloud service is as much about offloading complexity of administration as it is cost. With IaaS and PaaS, Microsoft administers the underlying infrastructure and automatically replicates all data to provide disaster recovery, configures and upgrades the database software, manages load balancing, and does transparent failover if there is a server failure within a data center.

- With **SQL databases and SQL managed instances**, you can continue to administer your database, but you no longer need to manage the database engine, the operating system, or the hardware. Examples of items you can continue to administer include databases and logins, index and query tuning, and auditing and security. Additionally, configuring high availability to another data center requires minimal configuration and administration.
- With **SQL virtual machines**, you have full control over the operating system and SQL Server instance configuration. With a VM, it's up to you to decide when to update/upgrade the operating system and database software and when to install any additional software such as anti-virus. Some automated features are provided to dramatically simplify patching, backup, and high availability. In addition, you can control the size of the VM, the number of disks, and their storage configurations. Azure allows you to change the size of a VM as needed. For information, see [Virtual Machine and Cloud Service Sizes for Azure](#).

Service Level Agreement (SLA)

For many IT departments, meeting up-time obligations of a Service Level Agreement (SLA) is a top priority. In this section, we look at what SLA applies to each database hosting option.

For **SQL Database**, Microsoft provides an availability SLA of 99.99%. For the latest information, see [Service Level Agreement](#).

For **SQL virtual machines**, Microsoft provides an availability SLA of 99.95% that covers just the Virtual Machine. This SLA does not cover the processes (such as SQL Server) running on the VM and requires that you host at least two VM instances in an availability set. For the latest information, see the [VM SLA](#). For database high availability (HA) within VMs, you should configure one of the supported high availability options in SQL Server, such as [Always On Availability Groups](#). Using a supported high availability option doesn't provide an additional SLA, but allows you to achieve >99.99% database availability.

Time to move to Azure

SQL database (single databases or elastic pools) are the right solution for cloud-designed applications when developer productivity and fast time-to-market for new solutions are critical. With programmatic DBA-like functionality, it is perfect for cloud architects and developers as it lowers the need for managing the underlying operating system and database.

SQL managed instance greatly simplifies the migration of existing applications to Azure SQL, enabling you to bring migrated database applications to market in Azure quickly.

SQL virtual machines are perfect if your existing or new applications require large databases or access to all features in SQL Server or Windows/Linux, and you want to avoid the time and expense of acquiring new on-premises hardware. It is also a good fit when you want to migrate existing on-premises applications and databases to Azure as-is - in cases where Azure SQL Database managed instance is not a good fit. Since you do not need to change the presentation, application, and data layers, you save time and budget on re-architecting your existing solution. Instead, you can focus on migrating all your solutions to Azure and in doing some performance optimizations that may be required by the Azure platform. For more information, see [Performance Best Practices for SQL Server on Azure Virtual Machines](#).

Create and manage Azure SQL resources with the Azure portal

The Azure portal provides a single page where you can manage [all of your Azure SQL resources](#) including your SQL virtual machines.

To access the **Azure SQL resources** page, from the Azure portal menu, select **Azure SQL** or search for and select **Azure SQL** in any page.

NOTE

Azure SQL provides a quick and easy way to access all of your SQL databases, elastic pools, database servers, SQL managed instances, and SQL virtual machines. Azure SQL is not a service or resource.

To manage existing resources, select the desired item in the list. To create new Azure SQL resources, select **+ Add**.

The screenshot shows the Microsoft Azure portal interface for the Azure SQL service. At the top, there's a search bar and a navigation bar with 'Microsoft Azure' and 'All services > Azure SQL'. Below this, the 'Azure SQL' service page is displayed. A red box highlights the '+ Add' button in the top-left corner of the main content area. The page lists two resources: 'contoso-database (contoso-sq...' and 'contoso-sql-server'. There are filters at the top for 'Subscription == all', 'Resource group == all', 'Location == all', and a 'No grouping' dropdown. At the bottom, there are navigation links for '< Previous', 'Page 1 of 1', and 'Next >'.

After selecting **+ Add**, view additional information about the different options by selecting **Show details** on any tile.

The screenshot shows the 'Select SQL deployment option' page in the Microsoft Azure portal. The title is 'Select SQL deployment option' with 'Microsoft' underneath. Below it is a 'Feedback' link. The page features three tiles: 'SQL databases', 'SQL managed instances', and 'SQL virtual machines'. The 'SQL databases' tile is highlighted with a dashed blue border. It contains the text: 'Best for modern cloud applications. Hyperscale and serverless options are available.' A 'Resource type' dropdown is set to 'Single database'. Below it are 'Create' and 'Show details' buttons. The other two tiles have similar structures with their own descriptions and resource type dropdowns. At the bottom, there are navigation links for '< Previous', 'Page 1 of 1', and 'Next >'.

For details, see:

- [Create a single database](#)
- [Create an elastic pool](#)
- [Create a managed instance](#)

- [Create a SQL virtual machine](#)

Next steps

- See [Your first Azure SQL database](#) to get started with SQL Database.
- See [SQL Database pricing](#).
- See [Provision a SQL Server virtual machine in Azure](#) to get started with SQL Server on Azure VMs.
- [Identify the right SQL database or SQL managed instance SKU for your on-premises database](#).

SQL Database release notes

11/20/2019 • 3 minutes to read • [Edit Online](#)

This article lists SQL Database features that are currently in public preview. For SQL Database updates and improvements, see [SQL Database service updates](#). For updates and improvements to other Azure services, see [Service updates](#).

Features in public preview

- [Single database](#)
- [Managed Instance](#)

FEATURE	DETAILS
New Fsv2-series and M-series hardware generations	For information, see Hardware generations .
Azure private link	Private Link simplifies the network architecture and secures the connection between endpoints in Azure by keeping data on the Azure network, thus eliminating exposure to the internet. Private Link also enables you to create and render your own services on Azure.
Accelerated database recovery with single databases and elastic pools	For information, see Accelerated Database Recovery .
Approximate Count Distinct	For information, see Approximate Count Distinct .
Batch Mode on Rowstore (under compatibility level 150)	For information, see Batch Mode on Rowstore .
Data discovery & classification	For information, see Azure SQL Database and SQL Data Warehouse data discovery & classification .
Elastic database jobs	For information, see Create, configure, and manage elastic jobs .
Elastic queries	For information, see Elastic query overview .
Elastic transactions	Distributed transactions across cloud databases .
Memory Grant Feedback (Row Mode) (under compatibility level 150)	For information, see Memory Grant Feedback (Row Mode) .
Query editor in the Azure portal	For information, see Use the Azure portal's SQL query editor to connect and query data .
R services / machine learning with single databases and elastic pools	For information, see Machine Learning Services in Azure SQL Database .
SQL Analytics	For information, see Azure SQL Analytics .

FEATURE	DETAILS
Table Variable Deferred Compilation (under compatibility level 150)	For information, see Table Variable Deferred Compilation .

New features

Managed instance H2 2019 updates

- [Auto-failover groups](#) enable you to replicate all databases from the primary instance to a secondary instance in another region.
- Configure your Managed instance behavior with [Global trace flags](#).

Managed instance H1 2019 updates

The following features are enabled in Managed instance deployment model in H1 2019:

- Support for subscriptions with [Azure monthly credit for Visual Studio subscribers](#) and increased [regional limits](#).
- Support for [SharePoint 2016](#) and [SharePoint 2019](#) and [Dynamics 365 Business Central](#)
- Create instances with [server-level collation](#) and [time-zone](#) of your choice.
- Managed instances are now protected with [built-in firewall](#).
- Configure instances to use [public endpoints](#), [Proxy override](#) connection to get better network performance, [4 vCores on Gen5 hardware generation](#) or [Configure backup retention up to 35 days](#) for Point-in-time restore. Long-term backup retention (up to 10 years) is still not enabled so you can use [Copy-only backups](#) as an alternative.
- New functionalities enable you to [geo-restore your database to another data center using PowerShell](#), [rename database](#), [delete virtual cluster](#).
- New built-in [Instance Contributor role](#) enables separation of duty (SoD) compliance with security principles and compliance with enterprise standards.
- Managed instance is available in the following Azure Government regions to GA (US Gov Texas, US Gov Arizona) as well as in China North 2 and China East 2. It is also available in the following public regions: Australia Central, Australia Central 2, Brazil South, France South, UAE Central, UAE North, South Africa North, South Africa West.

Fixed known issues

- **Aug 2019** - Contained databases are fully supported in managed instance.
- **Oct 2019** - Built-in point-in-time database restore from Business Critical tier to General Purpose tier will not succeed if source database contains in-memory OLTP objects.
- **Oct 2019** - You can use **Database Mail** feature with external (non-Azure) mail servers using secure connection.
- **Nov 2019** - Database consistency is verified using `DBCC CHECKDB` after restore database from Azure Blob Storage.

Updates

For a list of SQL Database updates and improvements, see [SQL Database service updates](#).

For updates and improvements to all Azure services, see [Service updates](#).

Contribute to content

To contribute to the Azure SQL Database documentation, see the [Docs Contributor Guide](#).

Azure SQL Database traffic migration to newer Gateways

11/7/2019 • 2 minutes to read • [Edit Online](#)

As Azure infrastructure improves, Microsoft will periodically refresh hardware to ensure we provide the best possible customer experience. In the coming months, we plan to add Gateways built on newer hardware generations, migrate traffic to them, and eventually decommission Gateways built on older hardware in some regions.

Customers will be notified via email and in the Azure portal well in advance of any change to Gateways available in each region. The most up-to-date information will be maintained in the [Azure SQL Database gateway IP addresses](#) table.

Impact of this change

The first round of traffic migration to newer Gateways is scheduled for **October 14, 2019** in the following regions:

- Brazil South
- West US
- West Europe
- East US
- Central US
- South East Asia
- South Central US
- North Europe
- North Central US
- Japan West
- Japan East
- East US 2
- East Asia

The traffic migration will change the public IP address that DNS resolves for your SQL Database. You will be impacted if you have:

- Hard coded the IP address for any particular Gateway in your on-premises firewall
- Any subnets using Microsoft.SQL as a Service Endpoint but cannot communicate with the Gateway IP addresses

You will not be impacted if you have :

- Redirection as the connection policy
- Connections to SQL Database from inside Azure and using Service Tags
- Connections made using supported versions of JDBC Driver for SQL Server will see no impact. For supported JDBC versions, see [Download Microsoft JDBC Driver for SQL Server](#).

What to do if you're affected

We recommend that you allow outbound traffic to IP addresses for all the [Azure SQL Database gateway IP addresses](#) in the region on TCP port 1433, and port range 11000-11999. This recommendation is applicable to

clients connecting from on-premises and also those connecting via Service Endpoints. For more information on port ranges, see [Connection policy](#).

Connections made from applications using Microsoft JDBC Driver below version 4.0 might fail certificate validation. Lower versions of Microsoft JDBC rely on Common Name (CN) in the Subject field of the certificate. The mitigation is to ensure that the hostNameInCertificate property is set to *.database.windows.net. For more information on how to set the hostNameInCertificate property, see [Connecting with SSL Encryption](#).

If the above mitigation doesn't work, file a support request for SQL Database using the following URL:
<https://aka.ms/getazuresupport>

Next steps

- Find out more about [Azure SQL Connectivity Architecture](#)

Planning for Azure maintenance events in Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

Learn how to prepare for planned maintenance events on your Azure SQL database.

What is a planned maintenance event

For each database, Azure SQL DB maintains a quorum of database replicas where one replica is the primary. At all times a primary replica must be online servicing, and at least one secondary replica must be healthy. During planned maintenance, members of the database quorum will go offline one at a time, with the intent that there is one responding primary replica and at least one secondary replica online to ensure no client downtime. When the primary replica needs to be brought offline, a reconfiguration/failover process will occur in which one secondary replica will become the new primary.

What to expect during a planned maintenance event

Reconfigurations/failovers generally complete within 30 seconds – the average is 8 seconds. If already connected, your application must reconnect to the healthy copy new primary replica of your database. If a new connection is attempted while the database is undergoing a reconfiguration before the new primary replica is online, you get error 40613 (Database Unavailable): "Database '{databasename}' on server '{servername}' is not currently available. Please retry the connection later.". If your database has a long running query, this query will be interrupted during a reconfiguration and will need to be restarted.

Retry Logic

Any client production application that connects to a cloud database service should implement a robust connection [retry logic](#). This will help mitigate these situations and should generally make the errors transparent to the end user.

Frequency

On average, 1.7 planned maintenance events occur each month.

Resource Health

If your SQL database is experiencing login failures, check the [Resource Health](#) window in the [Azure portal](#) for the current status. The Health History section contains the downtime reason for each event (when available).

Next steps

- Learn more about [Resource Health](#) for SQL Database
- For more information about retry logic, see [Retry logic for transient errors](#)

Choose between the vCore and the DTU purchasing models

2/24/2020 • 11 minutes to read • [Edit Online](#)

Azure SQL Database lets you easily purchase a fully managed platform as a service (PaaS) database engine that fits your performance and cost needs. Depending on the deployment model you've chosen for Azure SQL Database, you can select the purchasing model that works for you:

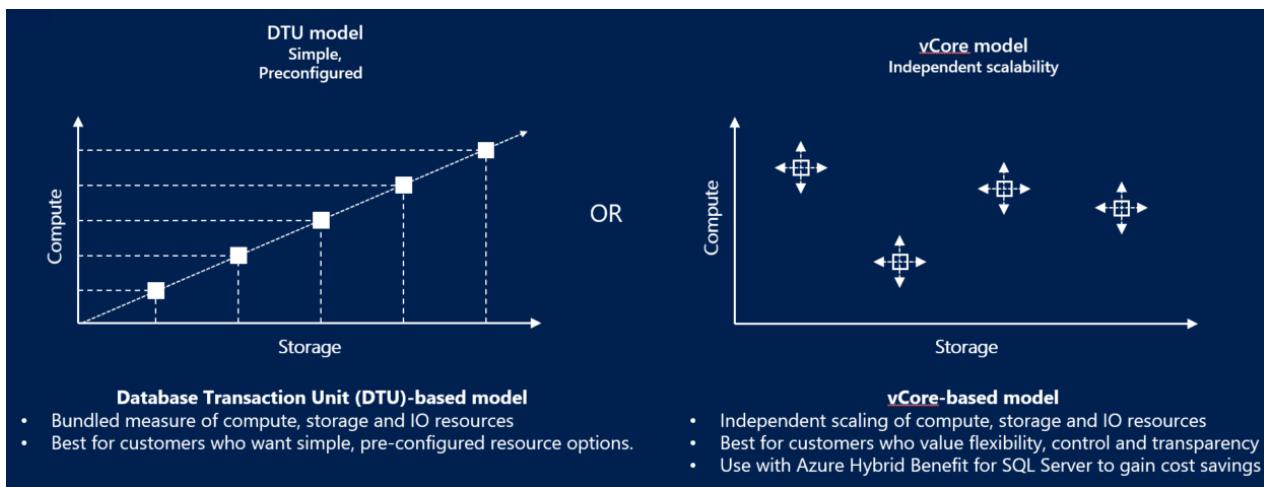
- **Virtual core (vCore)-based purchasing model** (recommended). This purchasing model provides a choice between a provisioned compute tier and a serverless compute tier. With the provisioned compute tier, you choose the exact amount of compute resources that are always provisioned for your workload. With the serverless compute tier, you specify the autoscaling of the compute resources over a configurable compute range. With this compute tier, you can also automatically pause and resume the database based on workload activity. The vCore unit price per unit of time is lower in the provisioned compute tier than it is in the serverless compute tier.
- **Database transaction unit (DTU)-based purchasing model**. This purchasing model provides bundled compute and storage packages balanced for common workloads.

Different purchasing models are available for different Azure SQL Database deployment models:

- The [single database](#) and [elastic pool](#) deployment options in [Azure SQL Database](#) offer both the [DTU-based purchasing model](#) and the [vCore-based purchasing model](#).
- The [managed instance](#) deployment option in Azure SQL Database offers only the [vCore-based purchasing model](#).
- The [Hyperscale service tier](#) is available for single databases that are using the [vCore-based purchasing model](#).

The following table and chart compare and contrast the vCore-based and the DTU-based purchasing models:

PURCHASING MODEL	DESCRIPTION	BEST FOR
DTU-based model	This model is based on a bundled measure of compute, storage, and I/O resources. Compute sizes are expressed in DTUs for single databases and in elastic database transaction units (eDTUs) for elastic pools. For more information about DTUs and eDTUs, see What are DTUs and eDTUs? .	Best for customers who want simple, preconfigured resource options.
vCore-based model	This model allows you to independently choose compute and storage resources. The vCore-based purchasing model also allows you to use Azure Hybrid Benefit for SQL Server to gain cost savings.	Best for customers who value flexibility, control, and transparency.



Compute costs

Provisioned compute costs

In the provisioned compute tier, the compute cost reflects the total compute capacity that is provisioned for the application.

In the Business Critical service tier, we automatically allocate at least 3 replicas. To reflect this additional allocation of compute resources, the price in the vCore-based purchasing model is approximately 2.7x higher in the Business Critical service tier than it is in the General Purpose service tier. Likewise, the higher storage price per GB in the Business Critical service tier reflects the higher IO limits and lower latency of the SSD storage.

The cost of backup storage is the same for the Business Critical service tier and the General Purpose service tier because both tiers use standard storage for backups.

Serverless compute costs

For a description of how compute capacity is defined and costs are calculated for the serverless compute tier, see [SQL Database serverless](#).

Storage costs

Different types of storage are billed differently. For data storage, you're charged for the provisioned storage based upon the maximum database or pool size you select. The cost doesn't change unless you reduce or increase that maximum. Backup storage is associated with automated backups of your instance and is allocated dynamically. Increasing your backup-retention period increases the backup storage that's consumed by your instance.

By default, 7 days of automated backups of your databases are copied to a read-access geo-redundant storage (RA-GRS) standard Blob storage account. This storage is used by weekly full backups, daily differential backups, and transaction log backups, which are copied every 5 minutes. The size of the transaction logs depends on the rate of change of the database. A minimum storage amount equal to 100 percent of the database size is provided at no extra charge. Additional consumption of backup storage is charged in GB per month.

For more information about storage prices, see the [pricing](#) page.

vCore-based purchasing model

A virtual core (vCore) represents a logical CPU and offers you the option to choose between generations of hardware and the physical characteristics of the hardware (for example, the number of cores, the memory, and the storage size). The vCore-based purchasing model gives you flexibility, control, transparency of individual resource consumption, and a straightforward way to translate on-premises workload requirements to the cloud. This model allows you to choose compute, memory, and storage resources based upon your workload needs.

In the vCore-based purchasing model, you can choose between the [General Purpose](#) and [Business Critical](#) service tiers for [single databases](#), [elastic pools](#), and [managed instances](#). For single databases, you can also choose the [Hyperscale service tier](#).

The vCore-based purchasing model lets you independently choose compute and storage resources, match on-premises performance, and optimize price. In the vCore-based purchasing model, you pay for:

- Compute resources (the service tier + the number of vCores and the amount of memory + the generation of hardware).
- The type and amount of data and log storage.
- Backup storage (RA-GRS).

IMPORTANT

Compute resources, I/O, and data and log storage are charged per database or elastic pool. Backup storage is charged per each database. For more information about managed instance charges, see [managed instances](#). **Region limitations:** For the current list of supported regions, see [products available by region](#). To create a managed instance in a region that currently isn't supported, [send a support request via the Azure portal](#).

If your single database or elastic pool consumes more than 300 DTUs, converting to the vCore-based purchasing model might reduce your costs. You can convert by using your API of choice or by using the Azure portal, with no downtime. However, conversion isn't required and isn't done automatically. If the DTU-based purchasing model meets your performance and business requirements, you should continue using it.

To convert from the DTU-based purchasing model to the vCore-based purchasing model, select the compute size by using the following rules of thumb:

- Every 100 DTUs in the standard tier require at least 1 vCore in the General Purpose service tier.
- Every 125 DTUs in the premium tier require at least 1 vCore in the Business Critical service tier.

DTU-based purchasing model

A database transaction unit (DTU) represents a blended measure of CPU, memory, reads, and writes. The DTU-based purchasing model offers a set of preconfigured bundles of compute resources and included storage to drive different levels of application performance. If you prefer the simplicity of a preconfigured bundle and fixed payments each month, the DTU-based model might be more suitable for your needs.

In the DTU-based purchasing model, you can choose between the basic, standard, and premium service tiers for both [single databases](#) and [elastic pools](#). The DTU-based purchasing model isn't available for [managed instances](#).

Database transaction units (DTUs)

For a single database at a specific compute size within a [service tier](#), Microsoft guarantees a certain level of resources for that database (independent of any other database in the Azure cloud). This guarantee provides a predictable level of performance. The amount of resources allocated for a database is calculated as a number of DTUs and is a bundled measure of compute, storage, and I/O resources.

The ratio among these resources is originally determined by an [online transaction processing \(OLTP\) benchmark workload](#) designed to be typical of real-world OLTP workloads. When your workload exceeds the amount of any of these resources, your throughput is throttled, resulting in slower performance and time-outs.

The resources used by your workload don't impact the resources available to other SQL databases in the Azure cloud. Likewise, the resources used by other workloads don't impact the resources available to your SQL database.

Database Transaction Unit – DTU

Bounding box

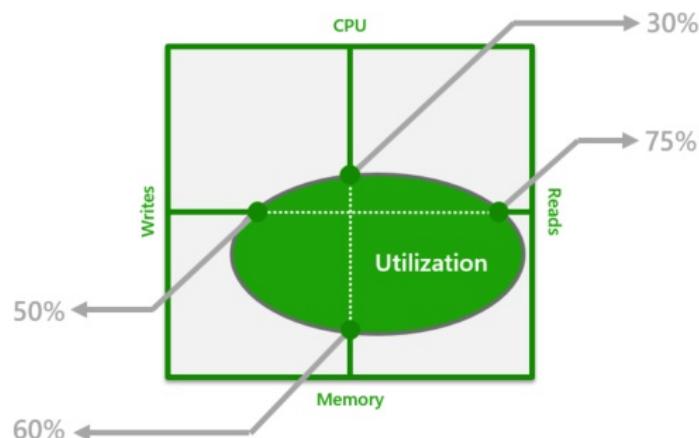
Monitoring database workload utilization within bounding box

Represents the relative power (resources) assigned to the database

Blended measure of CPU, memory, and read-write rates

Compare the power across performance levels

Simplifies talking about performance, think IOPS vs. %



DTUs are most useful for understanding the relative resources that are allocated for Azure SQL databases at different compute sizes and service tiers. For example:

- Doubling the DTUs by increasing the compute size of a database equates to doubling the set of resources available to that database.
- A premium service tier P11 database with 1750 DTUs provides 350x more DTU compute power than a basic service tier database with 5 DTUs.

To gain deeper insight into the resource (DTU) consumption of your workload, use [query-performance insights](#) to:

- Identify the top queries by CPU/duration/execution count that can potentially be tuned for improved performance. For example, an I/O-intensive query might benefit from [in-memory optimization techniques](#) to make better use of the available memory at a certain service tier and compute size.
- Drill down into the details of a query to view its text and its history of resource usage.
- Access performance-tuning recommendations that show actions taken by [SQL Database Advisor](#).

Elastic database transaction units (eDTUs)

For SQL databases that are always available, rather than provide a dedicated set of resources (DTUs) that might not always be needed, you can place these databases into an [elastic pool](#). The databases in an elastic pool are on a single Azure SQL Database server and share a pool of resources.

The shared resources in an elastic pool are measured by elastic database transaction units (eDTUs). Elastic pools provide a simple, cost-effective solution to manage performance goals for multiple databases that have widely varying and unpredictable usage patterns. An elastic pool guarantees that all the resources can't be consumed by one database in the pool, while ensuring that each database in the pool always has a minimum amount of necessary resources available.

A pool is given a set number of eDTUs for a set price. In the elastic pool, individual databases can autoscale within the configured boundaries. A database under a heavier load will consume more eDTUs to meet demand.

Databases under lighter loads will consume fewer eDTUs. Databases with no load will consume no eDTUs.

Because resources are provisioned for the entire pool, rather than per database, elastic pools simplify your management tasks and provide a predictable budget for the pool.

You can add additional eDTUs to an existing pool with no database downtime and with no impact on the databases in the pool. Similarly, if you no longer need extra eDTUs, remove them from an existing pool at any time. You can also add databases to or subtract databases from a pool at any time. To reserve eDTUs for other databases, limit the number of eDTUs a database can use under a heavy load. If a database consistently underuses resources, move it out of the pool and configure it as a single database with a predictable amount of required resources.

Determine the number of DTUs needed by a workload

If you want to migrate an existing on-premises or SQL Server virtual machine workload to Azure SQL Database, use the [DTU calculator](#) to approximate the number of DTUs needed. For an existing Azure SQL Database workload, use [query-performance insights](#) to understand your database-resource consumption (DTUs) and gain deeper insights for optimizing your workload. The [sys.dm_db_resource_stats](#) dynamic management view (DMV) lets you view resource consumption for the last hour. The [sys.resource_stats](#) catalog view displays resource consumption for the last 14 days, but at a lower fidelity of five-minute averages.

Determine DTU utilization

To determine the average percentage of DTU/eDTU utilization relative to the DTU/eDTU limit of a database or an elastic pool, use the following formula:

```
avg_dtu_percent = MAX(avg_cpu_percent, avg_data_io_percent, avg_log_write_percent)
```

The input values for this formula can be obtained from [sys.dm_db_resource_stats](#), [sys.resource_stats](#), and [sys.elastic_pool_resource_stats](#) DMVs. In other words, to determine the percentage of DTU/eDTU utilization toward the DTU/eDTU limit of a database or an elastic pool, pick the largest percentage value from the following: `avg_cpu_percent`, `avg_data_io_percent`, and `avg_log_write_percent` at a given point in time.

NOTE

The DTU limit of a database is determined by CPU, reads, writes, and memory available to the database. However, because the SQL Server database engine typically uses all available memory for its data cache to improve performance, the `avg_memory_usage_percent` value will usually be close to 100% regardless of current database load. Therefore, even though memory does indirectly influence the DTU limit, it is not used in the DTU utilization formula.

Workloads that benefit from an elastic pool of resources

Pools are well-suited for databases with a low resource-utilization average and relatively infrequent utilization spikes. For more information, see [When should you consider a SQL Database elastic pool?](#).

Frequently asked questions (FAQs)

Do I need to take my application offline to convert from a DTU-based service tier to a vCore-based service tier?

No. You don't need to take the application offline. The new service tiers offer a simple online-conversion method that's similar to the existing process of upgrading databases from the standard to the premium service tier and the other way around. You can start this conversion by using the Azure portal, PowerShell, the Azure CLI, T-SQL, or the REST API. See [Manage single databases](#) and [Manage elastic pools](#).

Can I convert a database from a service tier in the vCore-based purchasing model to a service tier in the DTU-based purchasing model?

Yes, you can easily convert your database to any supported performance objective by using the Azure portal, PowerShell, the Azure CLI, T-SQL, or the REST API. See [Manage single databases](#) and [Manage elastic pools](#).

Next steps

- For more information about the vCore-based purchasing model, see [vCore-based purchasing model](#).
- For more information about the DTU-based purchasing model, see [DTU-based purchasing model](#).

vCore model overview

2/26/2020 • 7 minutes to read • [Edit Online](#)

The virtual core (vCore) model provides several benefits:

- Higher compute, memory, IO, and storage limits.
- Control over the hardware generation to better match compute and memory requirements of the workload.
- Pricing discounts for [Azure Hybrid Benefit \(AHB\)](#) and [Reserved Instance \(RI\)](#).
- Greater transparency in the hardware details that power the compute; facilitates planning for migrations from on-premises deployments.

Service tiers

Service tier options in the vCore model include General Purpose, Business Critical, and Hyperscale. The service tier generally defines the storage architecture, space and IO limits, and business continuity options related to availability and disaster recovery.

	GENERAL PURPOSE	BUSINESS CRITICAL	HYPERSCALE
Best for	Most business workloads. Offers budget-oriented, balanced, and scalable compute and storage options.	Offers business applications the highest resilience to failures by using several isolated replicas, and provides the highest I/O performance per database replica.	Most business workloads with highly scalable storage and read-scale requirements. Offers higher resilience to failures by allowing configuration of more than one isolated database replica.
Storage	Uses remote storage. Single databases and elastic pools provisioned compute: 5 GB – 4 TB Serverless compute: 5 GB - 3 TB Managed Instance: 32 GB - 8 TB	Uses local SSD storage. Single databases and elastic pools provisioned compute: 5 GB – 4 TB Managed Instance: 32 GB - 4 TB	Flexible autogrow of storage as needed. Supports up to 100 TB of storage. Uses local SSD storage for local buffer-pool cache and local data storage. Uses Azure remote storage as final long-term data store.
IOPS and throughput (approximate)	Single databases and elastic pools: See resource limits for single databases and elastic pools . Managed Instance: See Overview Azure SQL Database managed instance resource limits .	See resource limits for single databases and elastic pools .	Hyperscale is a multi-tiered architecture with caching at multiple levels. Effective IOPS and throughput will depend on the workload.
Availability	1 replica, no read-scale replicas	3 replicas, 1 read-scale replica , zone-redundant high availability (HA)	1 read-write replica, plus 0-4 read-scale replicas

	GENERAL PURPOSE	BUSINESS CRITICAL	HYPERSCALE
Backups	Read-access geo-redundant storage (RA-GRS), 7-35 days (7 days by default)	RA-GRS, 7-35 days (7 days by default)	Snapshot-based backups in Azure remote storage. Restores use these snapshots for fast recovery. Backups are instantaneous and don't impact compute I/O performance. Restores are fast and aren't a size-of-data operation (taking minutes rather than hours or days).
In-memory	Not supported	Supported	Not supported

Choosing a service tier

For information on selecting a service tier for your particular workload, see the following articles:

- [When to choose the General purpose service tier](#)
- [When to choose the Business Critical service tier](#)
- [When to choose the Hyperscale service tier](#)

Compute tiers

Compute tier options in the vCore model include the provisioned and serverless compute tiers.

Provisioned compute

The provisioned compute tier provides a specific amount of compute resources that are continuously provisioned independent of workload activity, and bills for the amount of compute provisioned at a fixed price per hour.

Serverless compute

The [serverless compute tier](#) auto-scales compute resources based on workload activity, and bills for the amount of compute used per second.

Hardware generations

Hardware generation options in the vCore model include Gen 4/5, M-series (preview), and Fsv2-series (preview). The hardware generation generally defines the compute and memory limits and other characteristics that impact the performance of the workload.

Gen4/Gen5

- Gen4/Gen5 hardware provides balanced compute and memory resources, and is suitable for most database workloads that do not have higher memory, higher vCore, or faster single vCore requirements as provided by Fsv2-series or M-series.

For regions where Gen4/Gen5 is available, see [Gen4/Gen5 availability](#).

Fsv2-series(preview)

- Fsv2-series is a compute optimized hardware option delivering low CPU latency and high clock speed for the most CPU demanding workloads.
- Depending on the workload, Fsv2-series can deliver more CPU performance per vCore than Gen5, and the 72 vCore size can provide more CPU performance for less cost than 80 vCores on Gen5.

- Fsv2 provides less memory and tempdb per vCore than other hardware so workloads sensitive to those limits may want to consider Gen5 or M-series instead.

For regions where Fsv2-series is available, see [Fsv2-series availability](#).

M-series(preview)

- M-series is a memory optimized hardware option for workloads demanding more memory and higher compute limits than provided by Gen5.
- M-series provides 29 GB per vCore and 128 vCores, which increases the memory limit relative to Gen5 by 8x to nearly 4 TB.

To enable M-series hardware for a subscription and region, a support request must be opened. The subscription must be a paid offer type including Pay-As-You-Go or Enterprise Agreement (EA). If the support request is approved, then the selection and provisioning experience of M-series follows the same pattern as for other hardware generations. For regions where M-series is available, see [M-series availability](#).

Compute and memory specifications

HARDWARE GENERATION	COMPUTE	MEMORY
Gen4	<ul style="list-style-type: none"> - Intel E5-2673 v3 (Haswell) 2.4 GHz processors - Provision up to 24 vCores (1 vCore = 1 physical core) 	<ul style="list-style-type: none"> - 7 GB per vCore - Provision up to 168 GB
Gen5	<p>Provisioned compute</p> <ul style="list-style-type: none"> - Intel E5-2673 v4 (Broadwell) 2.3-GHz and Intel SP-8160 (Skylake) processors - Provision up to 80 vCores (1 vCore = 1 hyper-thread) <p>Serverless compute</p> <ul style="list-style-type: none"> - Intel E5-2673 v4 (Broadwell) 2.3-GHz and Intel SP-8160 (Skylake) processors - Auto-scale up to 16 vCores (1 vCore = 1 hyper-thread) 	<p>Provisioned compute</p> <ul style="list-style-type: none"> - 5.1 GB per vCore - Provision up to 408 GB <p>Serverless compute</p> <ul style="list-style-type: none"> - Auto-scale up to 24 GB per vCore - Auto-scale up to 48 GB max
Fsv2-series	<ul style="list-style-type: none"> - Intel Xeon Platinum 8168 (SkyLake) processors - Featuring a sustained all core turbo clock speed of 3.4 GHz and a maximum single core turbo clock speed of 3.7 GHz. - Provision 72 vCores (1 vCore = 1 hyper-thread) 	<ul style="list-style-type: none"> - 1.9 GB per vCore - Provision 136 GB
M-series	<ul style="list-style-type: none"> - Intel Xeon E7-8890 v3 2.5 GHz processors - Provision 128 vCores (1 vCore = 1 hyper-thread) 	<ul style="list-style-type: none"> - 29 GB per vCore - Provision 3.7 TB

For more information on resource limits, see [Resource limits for single databases \(vCore\)](#), or [Resource limits for elastic pools \(vCore\)](#).

Selecting a hardware generation

In the Azure portal, you can select the hardware generation for a SQL database or pool at the time of creation, or you can change the hardware generation of an existing SQL database or pool.

To select a hardware generation when creating a SQL database or pool

For detailed information, see [Create a SQL database](#).

On the **Basics** tab, select the **Configure database** link in the **Compute + storage** section, and then select the **Change configuration** link:

Home > Create SQL Database > Configure

Configure

Feedback

Looking for basic, standard, premium?

General Purpose
Scalable compute and storage options
Up to 7,000 IOPS
5-10 ms latency

Hyperscale
On-demand scalable storage
Data up to 200,000 IOPS, 1-2 ms latency
Log up to 7,000 IOPS, 5-10ms latency

Business Critical
High transaction rate and high resiliency
Up to 200,000 IOPS
1-2 ms latency

Compute tier

Provisioned
Compute resources are pre-allocated
Billed per hour based on vCores configured

Serverless
Compute resources are auto-scaled
Billed per second based on vCores used

Compute Hardware

Select a hardware generation best suited to your application. Select from General purpose, compute-optimized and memory optimized compute.

Hardware Configuration

Gen5
up to 80 vCores, up to 408 GB memory
Change configuration

Save money

Save up to 55% with a license you already own. Already have a SQL Server license? Yes No

vCores [How do vCores compare with DTUs?](#)

2 vCores

Data max size 32 GB 1 TB

9.6 GB LOG SPACE ALLOCATED

Cost summary

Gen5 - General Purpose (GP_Gen5_2)
Cost per vCore (in USD)
vCores selected x 2

Cost per GB (in USD)
Max storage selected (in GB)

ESTIMATED COST / MONTH

Apply

Select the desired hardware generation:

Home > SQL databases > Create SQL Database > Configure > SQL hardware configuration

SQL hardware configuration

SQL database

Available hardware configurations

Based on your workload requirements, select from available hardware configurations listed below.

Configuration	Description	Max vCores	Max memory	Max storage
Gen4	Balanced memory and compute	24	168 GB	4 TB
Gen5	Balanced memory and compute	80	408 GB	4 TB
Fsv2 Series	Compute optimized	72	136 GB	4 TB
M-Series	Memory optimized	128	3.68 TB	4 TB

OK Cancel

To change the hardware generation of an existing SQL database or pool

For a database, on the Overview page, select the **Pricing tier** link:

Home > adventure-works (102119091027/adventure-works)

adventure-works (102119091027/adventure-works) SQL database

Search (Ctrl+ /) Copy Restore Export Set server firewall Delete Connect with... Feedback

Overview Resource group (change) : 102119091027-rg

Status : Online

Location : West US 2

Subscription (change) : SQL

Subscription ID :

Tags (change) : Click here to add tags

Server name : 102119091027.database.windows.net

Elastic pool : No elastic pool

Connection strings : Show database connection strings

Pricing tier : General Purpose: Gen4, 1 vCore

Earliest restore point : 2019-10-21 16:46 UTC

For a pool, on the Overview page, select **Configure**.

Follow the steps to change configuration, and select the hardware generation as described in the previous steps.

To select a hardware generation when creating a managed instance

For detailed information, see [Create a managed instance](#).

On the **Basics** tab, select the **Configure database** link in the **Compute + storage** section, and then select desired hardware generation:

Home > SQL managed instances > Create Azure SQL Database Managed Instance > Configure performance

Configure performance

Feedback

General Purpose

For most production workloads
4 / 8 / 16 / 24 / 32 / 40 / 64 / 80 vCores
32 GB - 8 TB storage capacity
Fast storage

Business Critical

For IO-intensive and compute-intensive workloads.
4 / 8 / 16 / 24 / 32 / 40 / 64 / 80 vCores
32 GB - 4 TB storage capacity
Super fast storage

Compute Generation

Hardware generation is managed and upgraded periodically. [Learn more](#)

Gen4 Gen5

vCores [What is a vCore?](#)

8

Storage in GB. Custom amount will be rounded to the nearest value divisible by 32.

256

Backup

Backup storage, up to the same amount of 256 GB as data storage, is provided free of charge. Consuming backup storage above this amount is billed per usage. [Learn more](#)

Save money [①](#)

Save up to 55% with a license you already own. Already have a SQL Server license? [Yes](#) [No](#)

By selecting "Yes", I confirm that I have a SQL Server license with Software Assurance to apply this Azure Hybrid Benefit for SQL Server. [Learn more](#)

Monthly cost

To change the hardware generation of an existing managed instance

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

From the managed instance page, select **Pricing tier** link placed under the Settings section

Home > SQL managed instances > adventure-works

adventure-works
SQL managed instance

Search (Ctrl+ /) <>

New database Reset password Delete Feedback

✓ Your Managed Instance is ready. Click here to get started.

Resource group (change) : resource-group
Status : Online
Location : West Europe
Subscription (change) :
Subscription ID :
Tags (change) : Click here to add tags

CPU utilization

100%
90%
80%
70%
60%

Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Pricing tier
Connection strings
Active Directory admin
Instance Failover Groups
Deleted databases

On the **Pricing tier** page you will be able to change hardware generation as described in the previous steps.

Hardware availability

Gen4/Gen5

Gen4 hardware is [being phased out](#) and is not available anymore for the new deployments. All new databases must be deployed on Gen5 hardware.

Gen5 is available in most regions worldwide.

Fsv2-series

Fsv2-series is available in the following regions: Australia Central, Australia Central 2, Australia East, Australia Southeast, Brazil South, Canada Central, East Asia, East Us, France Central, India Central, India West, Korea Central, Korea South, North Europe, South Africa North, Southeast Asia, UK South, UK West, West Europe, West Us 2.

M-series

M-series is available in the following regions: East US, North Europe, West Europe, West US 2. M-series may also have limited availability in additional regions. You can request a different region than listed here, but fulfillment in a different region may not be possible.

To enable M-series availability in a subscription, access must be requested by [filing a new support request](#).

Create a support request to enable M-series:

1. Select **Help + support** in the portal.
2. Select **New support request**.

On the **Basics** page, provide the following:

1. For **Issue type**, select **Service and subscription limits (quotas)**.
2. For **Subscription** = select the subscription to enable M-series.
3. For **Quota type**, select **SQL database**.
4. Select **Next** to go to the **Details** page.

On the **Details** page, provide the following:

1. In the **PROBLEM DETAILS** section select the **Provide details** link.
2. For **SQL Database quota type** select **M-series**.
3. For **Region**, select the region to enable M-series. For regions where M-series is available, see [M-series availability](#).

Approved support requests are typically fulfilled within 5 business days.

Next steps

- To create a SQL database, see [Creating a SQL database using the Azure portal](#).
- For the specific compute sizes and storage size choices available for single databases, see [SQL Database vCore-based resource limits for single databases](#).
- For the specific compute sizes and storage size choices available for elastic pools, see [SQL Database vCore-based resource limits for elastic pools](#).
- For pricing details, see the [Azure SQL Database pricing page](#).

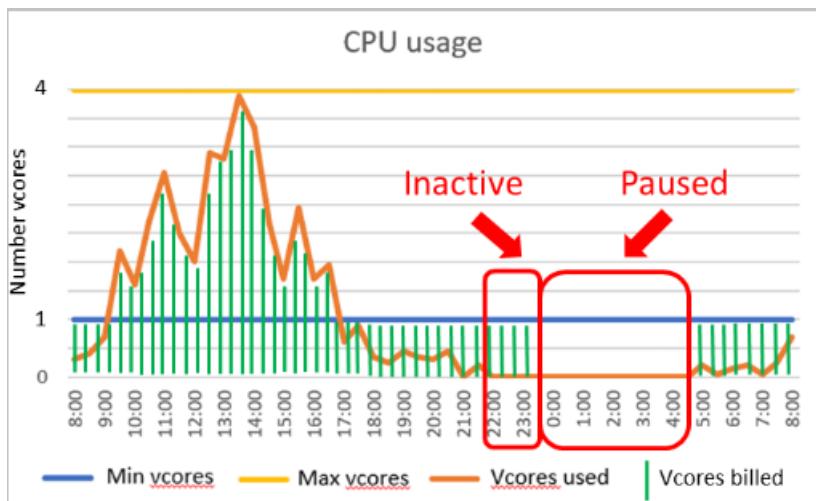
Azure SQL Database serverless

2/11/2020 • 14 minutes to read • [Edit Online](#)

Azure SQL Database serverless is a compute tier for single databases that automatically scales compute based on workload demand and bills for the amount of compute used per second. The serverless compute tier also automatically pauses databases during inactive periods when only storage is billed and automatically resumes databases when activity returns.

Serverless compute tier

The serverless compute tier for a single database is parameterized by a compute autoscaling range and an autopause delay. The configuration of these parameters shape the database performance experience and compute cost.



Performance configuration

- The **minimum vCores** and **maximum vCores** are configurable parameters that define the range of compute capacity available for the database. Memory and IO limits are proportional to the vCore range specified.
- The **autopause delay** is a configurable parameter that defines the period of time the database must be inactive before it is automatically paused. The database is automatically resumed when the next login or other activity occurs. Alternatively, autopauses can be disabled.

Cost

- The cost for a serverless database is the summation of the compute cost and storage cost.
- When compute usage is between the min and max limits configured, the compute cost is based on vCore and memory used.
- When compute usage is below the min limits configured, the compute cost is based on the min vCores and min memory configured.
- When the database is paused, the compute cost is zero and only storage costs are incurred.
- The storage cost is determined in the same way as in the provisioned compute tier.

For more cost details, see [Billing](#).

Scenarios

Serverless is price-performance optimized for single databases with intermittent, unpredictable usage patterns that can afford some delay in compute warm-up after idle usage periods. In contrast, the provisioned compute

tier is price-performance optimized for single databases or multiple databases in elastic pools with higher average usage that cannot afford any delay in compute warm-up.

Scenarios well-suited for serverless compute

- Single databases with intermittent, unpredictable usage patterns interspersed with periods of inactivity and lower average compute utilization over time.
- Single databases in the provisioned compute tier that are frequently rescaled and customers who prefer to delegate compute rescaling to the service.
- New single databases without usage history where compute sizing is difficult or not possible to estimate prior to deployment in SQL Database.

Scenarios well-suited for provisioned compute

- Single databases with more regular, predictable usage patterns and higher average compute utilization over time.
- Databases that cannot tolerate performance trade-offs resulting from more frequent memory trimming or delay in autoresuming from a paused state.
- Multiple databases with intermittent, unpredictable usage patterns that can be consolidated into elastic pools for better price-performance optimization.

Comparison with provisioned compute tier

The following table summarizes distinctions between the serverless compute tier and the provisioned compute tier:

	SERVERLESS COMPUTE	PROVISIONED COMPUTE
Database usage pattern	Intermittent, unpredictable usage with lower average compute utilization over time.	More regular usage patterns with higher average compute utilization over time, or multiple databases using elastic pools.
Performance management effort	Lower	Higher
Compute scaling	Automatic	Manual
Compute responsiveness	Lower after inactive periods	Immediate
Billing granularity	Per second	Per hour

Purchasing model and service tier

SQL Database serverless is currently only supported in the General Purpose tier on Generation 5 hardware in the vCore purchasing model.

Autoscaling

Scaling responsiveness

In general, serverless databases are run on a machine with sufficient capacity to satisfy resource demand without interruption for any amount of compute requested within limits set by the max vCores value. Occasionally, load balancing automatically occurs if the machine is unable to satisfy resource demand within a few minutes. For example, if the resource demand is 4 vCores, but only 2 vCores are available, then it may take up to a few minutes to load balance before 4 vCores are provided. The database remains online during load balancing except for a brief period at the end of the operation when connections are dropped.

Memory management

Memory for serverless databases is reclaimed more frequently than for provisioned compute databases. This behavior is important to control costs in serverless and can impact performance.

Cache reclamation

Unlike provisioned compute databases, memory from the SQL cache is reclaimed from a serverless database when CPU or cache utilization is low.

- Cache utilization is considered low when the total size of the most recently used cache entries falls below a threshold for a period of time.
- When cache reclamation is triggered, the target cache size is reduced incrementally to a fraction of its previous size and reclaiming only continues if usage remains low.
- When cache reclamation occurs, the policy for selecting cache entries to evict is the same selection policy as for provisioned compute databases when memory pressure is high.
- The cache size is never reduced below the min memory limit as defined by min vCores which can be configured.

In both serverless and provisioned compute databases, cache entries may be evicted if all available memory is used.

Cache hydration

The SQL cache grows as data is fetched from disk in the same way and with the same speed as for provisioned databases. When the database is busy, the cache is allowed to grow unconstrained up to the max memory limit.

Autopausing and autoresuming

Autopausing

Autopausing is triggered if all of the following conditions are true for the duration of the autopause delay:

- Number sessions = 0
- CPU = 0 for user workload running in the user pool

An option is provided to disable autopausing if desired.

The following features do not support autopausing. That is, if any of the following features are used, then the database remains online regardless of the duration of database inactivity:

- Geo-replication (active geo-replication and auto-failover groups).
- Long-term backup retention (LTR).
- The sync database used in SQL data sync. Unlike sync databases, hub and member databases support autopausing.
- The job database used in elastic jobs.

Autopausing is temporarily prevented during the deployment of some service updates which require the database be online. In such cases, autopausing becomes allowed again once the service update completes.

Autoresuming

Autoresuming is triggered if any of the following conditions are true at any time:

FEATURE	AUTORESUME TRIGGER
Authentication and authorization	Login

FEATURE	AUTORESUME TRIGGER
Threat detection	Enabling/disabling threat detection settings at the database or server level. Modifying threat detection settings at the database or server level.
Data discovery and classification	Adding, modifying, deleting, or viewing sensitivity labels
Auditing	Viewing auditing records. Updating or viewing auditing policy.
Data masking	Adding, modifying, deleting, or viewing data masking rules
Transparent data encryption	View state or status of transparent data encryption
Query (performance) data store	Modifying or viewing query store settings
Autotuning	Application and verification of autotuning recommendations such as auto-indexing
Database copying	Create database as copy. Export to a BACPAC file.
SQL data sync	Synchronization between hub and member databases that run on a configurable schedule or are performed manually
Modifying certain database metadata	Adding new database tags. Changing max vCores, min vCores, or autopause delay.
SQL Server Management Studio (SSMS)	Using SSMS versions earlier than 18.1 and opening a new query window for any database in the server will resume any auto-paused database in the same server. This behavior does not occur if using SSMS version 18.1 or later.

Autoresuming is also triggered during the deployment of some service updates which require the database be online.

Connectivity

If a serverless database is paused, then the first login will resume the database and return an error stating that the database is unavailable with error code 40613. Once the database is resumed, the login must be retried to establish connectivity. Database clients with connection retry logic should not need to be modified.

Latency

The latency to autoresume and autopause a serverless database is generally order of 1 minute to autoresume and 1-10 minutes to autopause.

Onboarding into serverless compute tier

Creating a new database or moving an existing database into a serverless compute tier follows the same pattern as creating a new database in provisioned compute tier and involves the following two steps.

1. Specify the service objective name. The service objective prescribes the service tier, hardware generation, and max vCores. The following table shows the service objective options:

Service Objective Name	Service Tier	Hardware Generation	Max vCores
GP_S_Gen5_1	General Purpose	Gen5	1
GP_S_Gen5_2	General Purpose	Gen5	2
GP_S_Gen5_4	General Purpose	Gen5	4
GP_S_Gen5_6	General Purpose	Gen5	6
GP_S_Gen5_8	General Purpose	Gen5	8
GP_S_Gen5_10	General Purpose	Gen5	10
GP_S_Gen5_12	General Purpose	Gen5	12
GP_S_Gen5_14	General Purpose	Gen5	14
GP_S_Gen5_16	General Purpose	Gen5	16

2. Optionally, specify the min vCores and autopause delay to change their default values. The following table shows the available values for these parameters.

Parameter	Value Choices	Default Value
Min vCores	Depends on max vCores configured - see resource limits .	0.5 vCores
Autopause delay	Minimum: 60 minutes (1 hour) Maximum: 10080 minutes (7 days) Increments: 60 minutes Disable autopause: -1	60 minutes

Create new database in serverless compute tier

The following examples create a new database in the serverless compute tier. The examples explicitly specify the min vCores, max vCores, and autopause delay.

Use Azure portal

See [Quickstart: Create a single database in Azure SQL Database using the Azure portal](#).

Use PowerShell

```
New-AzSqlDatabase -ResourceGroupName $resourceGroupName -ServerName $serverName -DatabaseName $databaseName ` 
-ComputeModel Serverless -Edition GeneralPurpose -ComputeGeneration Gen5 ` 
-MinVcore 0.5 -MaxVcore 2 -AutoPauseDelayInMinutes 720
```

Use Azure CLI

```
az sql db create -g $resourceGroupName -s $serverName -n $databaseName ` 
-e GeneralPurpose -f Gen5 -min-capacity 0.5 -c 2 --compute-model Serverless --auto-pause-delay 720
```

Use Transact-SQL (T-SQL)

The following example creates a new database in the serverless compute tier.

```
CREATE DATABASE testdb
( EDITION = 'GeneralPurpose', SERVICE_OBJECTIVE = 'GP_S_Gen5_1' ) ;
```

For details, see [CREATE DATABASE](#).

Move database from provisioned compute tier into serverless compute tier

The following examples move a database from the provisioned compute tier into the serverless compute tier. The examples explicitly specify the min vCores, max vCores, and autopause delay.

Use PowerShell

```
Set-AzSqlDatabase -ResourceGroupName $resourceGroupName -ServerName $serverName -DatabaseName $databaseName ` 
    -Edition GeneralPurpose -ComputeModel Serverless -ComputeGeneration Gen5 ` 
    -MinVcore 1 -MaxVcore 4 -AutoPauseDelayInMinutes 1440
```

Use Azure CLI

```
az sql db update -g $resourceGroupName -s $serverName -n $databaseName ` 
    --edition GeneralPurpose --min-capacity 1 --capacity 4 --family Gen5 --compute-model Serverless --auto-` 
    pause-delay 1440
```

Use Transact-SQL (T-SQL)

The following example moves a database from the provisioned compute tier into the serverless compute tier.

```
ALTER DATABASE testdb
MODIFY ( SERVICE_OBJECTIVE = 'GP_S_Gen5_1' ) ;
```

For details, see [ALTER DATABASE](#).

Move database from serverless compute tier into provisioned compute tier

A serverless database can be moved into a provisioned compute tier in the same way as moving a provisioned compute database into a serverless compute tier.

Modifying serverless configuration

Use PowerShell

Modifying the maximum or minimum vCores, and autopause delay, is performed by using the [Set-AzSqlDatabase](#) command in PowerShell using the `MaxVcore`, `MinVcore`, and `AutoPauseDelayInMinutes` arguments.

Use Azure CLI

Modifying the maximum or minimum vCores, and autopause delay, is performed by using the [az sql db update](#) command in Azure CLI using the `capacity`, `min-capacity`, and `auto-pause-delay` arguments.

Monitoring

Resources used and billed

The resources of a serverless database are encapsulated by app package, SQL instance, and user resource pool entities.

App package

The app package is the outer most resource management boundary for a database, regardless of whether the database is in a serverless or provisioned compute tier. The app package contains the SQL instance and external services that together scope all user and system resources used by a database in SQL Database. Examples of

external services include R and full-text search. The SQL instance generally dominates the overall resource utilization across the app package.

User resource pool

The user resource pool is the inner most resource management boundary for a database, regardless of whether the database is in a serverless or provisioned compute tier. The user resource pool scopes CPU and IO for user workload generated by DDL queries such as CREATE and ALTER and DML queries such as SELECT, INSERT, UPDATE, and DELETE. These queries generally represent the most substantial proportion of utilization within the app package.

Metrics

Metrics for monitoring the resource usage of the app package and user pool of a serverless database are listed in the following table:

ENTITY	METRIC	DESCRIPTION	UNITS
App package	app_cpu_percent	Percentage of vCores used by the app relative to max vCores allowed for the app.	Percentage
App package	app_cpu_billed	<p>The amount of compute billed for the app during the reporting period. The amount paid during this period is the product of this metric and the vCore unit price.</p> <p>Values of this metric are determined by aggregating over time the maximum of CPU used and memory used each second. If the amount used is less than the minimum amount provisioned as set by the min vCores and min memory, then the minimum amount provisioned is billed. In order to compare CPU with memory for billing purposes, memory is normalized into units of vCores by rescaling the amount of memory in GB by 3 GB per vCore.</p>	vCore seconds
App package	app_memory_percent	Percentage of memory used by the app relative to max memory allowed for the app.	Percentage
User pool	cpu_percent	Percentage of vCores used by user workload relative to max vCores allowed for user workload.	Percentage

ENTITY	METRIC	DESCRIPTION	UNITS
User pool	data_IO_percent	Percentage of data IOPS used by user workload relative to max data IOPS allowed for user workload.	Percentage
User pool	log_IO_percent	Percentage of log MB/s used by user workload relative to max log MB/s allowed for user workload.	Percentage
User pool	workers_percent	Percentage of workers used by user workload relative to max workers allowed for user workload.	Percentage
User pool	sessions_percent	Percentage of sessions used by user workload relative to max sessions allowed for user workload.	Percentage

Pause and resume status

In the Azure portal, the database status is displayed in the overview pane of the server that lists the databases it contains. The database status is also displayed in the overview pane for the database.

Using the following commands to query the pause and resume status of a database:

Use PowerShell

```
Get-AzSqlDatabase -ResourceGroupName $resourcegroupname -ServerName $servername -DatabaseName $databasename `| Select -ExpandProperty "Status"
```

Use Azure CLI

```
az sql db show --name $databasename --resource-group $resourcegroupname --server $servername --query 'status' -o json
```

Resource limits

For resource limits, see [serverless compute tier](#).

Billing

The amount of compute billed is the maximum of CPU used and memory used each second. If the amount of CPU used and memory used is less than the minimum amount provisioned for each, then the provisioned amount is billed. In order to compare CPU with memory for billing purposes, memory is normalized into units of vCores by rescaling the amount of memory in GB by 3 GB per vCore.

- **Resource billed:** CPU and memory
- **Amount billed:** vCore unit price * max (min vCores, vCores used, min memory GB * 1/3, memory GB used * 1/3)
- **Billing frequency:** Per second

The vCore unit price is the cost per vCore per second. Refer to the [Azure SQL Database pricing page](#) for specific unit prices in a given region.

The amount of compute billed is exposed by the following metric:

- **Metric:** app_cpu_billed (vCore seconds)
- **Definition:** max (min vCores, vCores used, min memory GB * 1/3, memory GB used * 1/3)
- **Reporting frequency:** Per minute

This quantity is calculated each second and aggregated over 1 minute.

Consider a serverless database configured with 1 min vCore and 4 max vCores. This corresponds to around 3 GB min memory and 12-GB max memory. Suppose the auto-pause delay is set to 6 hours and the database workload is active during the first 2 hours of a 24-hour period and otherwise inactive.

In this case, the database is billed for compute and storage during the first 8 hours. Even though the database is inactive starting after the second hour, it is still billed for compute in the subsequent 6 hours based on the minimum compute provisioned while the database is online. Only storage is billed during the remainder of the 24-hour period while the database is paused.

More precisely, the compute bill in this example is calculated as follows:

TIME INTERVAL	VCORES USED EACH SECOND	GB USED EACH SECOND	COMPUTE DIMENSION BILLED	VCORE SECONDS BILLED OVER TIME INTERVAL
0:00-1:00	4	9	vCores used	4 vCores * 3600 seconds = 14400 vCore seconds
1:00-2:00	1	12	Memory used	12 GB * 1/3 * 3600 seconds = 14400 vCore seconds
2:00-8:00	0	0	Min memory provisioned	3 GB * 1/3 * 21600 seconds = 21600 vCore seconds
8:00-24:00	0	0	No compute billed while paused	0 vCore seconds
Total vCore seconds billed over 24 hours				50400 vCore seconds

Suppose the compute unit price is \$0.000145/vCore/second. Then the compute billed for this 24-hour period is the product of the compute unit price and vCore seconds billed: $\$0.000145/\text{vCore/second} * 50400 \text{ vCore seconds} \sim \7.31

Azure Hybrid Benefit and reserved capacity

Azure Hybrid Benefit (AHB) and reserved capacity discounts do not apply to the serverless compute tier.

Available regions

The serverless compute tier is available worldwide except the following regions: China East, China North, Germany Central, Germany Northeast, UK North, UK South 2, West Central US, and US Gov Central (Iowa).

Next steps

- To get started, see [Quickstart: Create a single database in Azure SQL Database using the Azure portal](#).
- For resource limits, see [Serverless compute tier resource limits](#).

Azure Hybrid Benefit

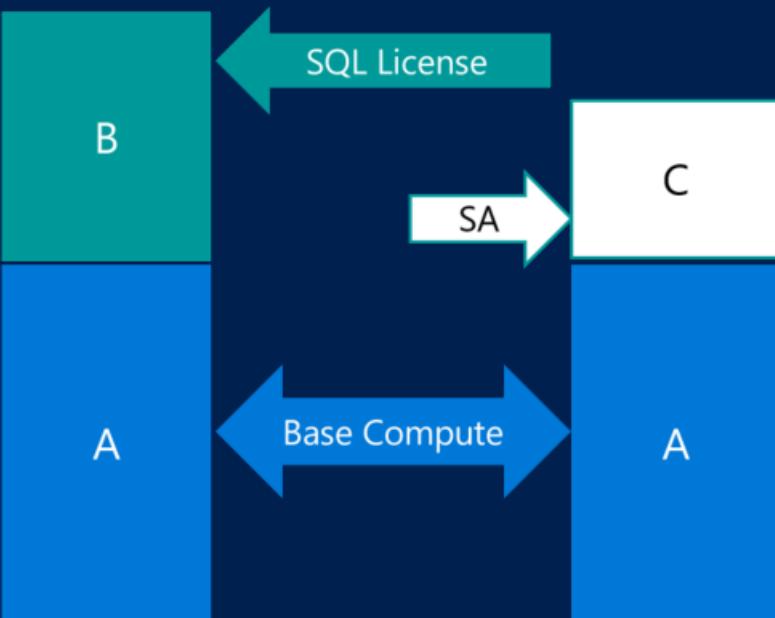
1/14/2020 • 2 minutes to read • [Edit Online](#)

In the provisioned compute tier of the vCore-based purchasing model, you can exchange your existing licenses for discounted rates on SQL Database by using [Azure Hybrid Benefit for SQL Server](#). This Azure benefit allows you to save up to 30 percent or even higher on Azure SQL Database by using your on-premises SQL Server licenses with Software Assurance. Please do use the Azure Hybrid Benefit Calculator using the link mentioned before for correct values.

NOTE

Changing to Azure Hybrid Benefit does not require any downtime.

SQL Database pricing structure in [vCore Resourcing Model](#)



Choose a license model

With Azure Hybrid Benefit, you can choose to pay only for the underlying Azure infrastructure by using your existing SQL Server license for the SQL database engine itself (Base Compute pricing), or you can pay for both the underlying infrastructure and the SQL Server license (License-Included pricing).

You can choose or change your licensing model by using the Azure portal or by using one of the following APIs:

- [PowerShell](#)
- [Azure CLI](#)
- [REST API](#)

To set or update the license type by using PowerShell:

- [New-AzSqlDatabase](#)
- [Set-AzSqlDatabase](#)
- [New-AzSqlInstance](#)
- [Set-AzSqlInstance](#)

Next steps

- For choosing between the SQL Database deployment options, see [Choose the right deployment option in Azure SQL](#).
- For a comparison of SQL Database features, see [Azure SQL Database Features](#).

Save costs for SQL Database compute resources with Azure SQL Database reserved capacity

1/14/2020 • 5 minutes to read • [Edit Online](#)

Save money with Azure SQL Database by committing to a reservation for compute resources compared to pay-as-you-go prices. With Azure SQL Database reserved capacity, you make a commitment for SQL Database use for a period of one or three years to get a significant discount on the compute costs. To purchase SQL Database reserved capacity, you need to specify the Azure region, deployment type, performance tier, and term.

You do not need to assign the reservation to specific SQL Database instances (single databases, elastic pools, or managed instances). Matching SQL Database instances, that are already running or ones that are newly deployed, will automatically get the benefit. By purchasing a reservation, you are commit to usage for the compute costs for a period of one or three years. As soon as you buy a reservation, the SQL Database compute charges that match the reservation attributes are no longer charged at the pay-as-you go rates. A reservation does not cover software, networking, or storage charges associated with the SQL Database instance. At the end of the reservation term, the billing benefit expires and the SQL Databases are billed at the pay-as-you go price. Reservations do not auto-renew. For pricing information, see the [SQL Database reserved capacity offering](#).

You can buy Azure SQL Database reserved capacity in the [Azure portal](#). Pay for the reservation [up front or with monthly payments](#). To buy SQL Database reserved capacity:

- You must be in the owner role for at least one Enterprise or individual subscription with pay-as-you-go rates.
- For Enterprise subscriptions, **Add Reserved Instances** must be enabled in the [EA portal](#). Or, if that setting is disabled, you must be an EA Admin on the subscription.
- For Cloud Solution Provider (CSP) program, only the admin agents or sales agents can purchase SQL Database reserved capacity.

The details on how enterprise customers and Pay-As-You-Go customers are charged for reservation purchases, see [understand Azure reservation usage for your Enterprise enrollment](#) and [understand Azure reservation usage for your Pay-As-You-Go subscription](#).

Determine the right SQL size before purchase

The size of reservation should be based on the total amount of compute used by the existing or soon-to-be-deployed single databases, elastic pools, or managed instances within a specific region and using the same performance tier and hardware generation.

For example, let's suppose that you are running one general purpose, Gen5 – 16 vCore elastic pool, and two business critical, Gen5 – 4 vCore single databases. Further, let's supposed that you plan to deploy within the next month an additional general purpose, Gen5 – 16 vCore elastic pool, and one business critical, Gen5 – 32 vCore elastic pool. Also, let's suppose that you know that you will need these resources for at least 1 year. In this case, you should purchase a 32 (2x16) vCores, 1 year reservation for single database/elastic pool general purpose - Gen5 and a 40 (2x4 + 32) vCore 1 year reservation for single database/elastic pool business critical - Gen5.

Buy SQL Database reserved capacity

1. Sign in to the [Azure portal](#).
2. Select **All services > Reservations**.
3. Select **Add** and then in the Purchase reservations pane, select **SQL Database** to purchase a new reservation for SQL Database.

4. Fill-in the required fields. Existing or new single databases, elastic pools, or managed instances that match the attributes you select qualify to get the reserved capacity discount. The actual number of your SQL Database instances that get the discount depend on the scope and quantity selected.

PERFORMANCE TIER	REGION	TERM	DEPLOYMENT TYPE
SQL Database Managed Instance Business Critical - Compute Gen4	West US 2	One Year	SQL Database Managed Instance
SQL Database Managed Instance Business Critical - Compute Gen5	West US 2	One Year	SQL Database Managed Instance
SQL Database Managed Instance General Purpose - Compute Gen4	West US 2	One Year	SQL Database Managed Instance
SQL Database Managed Instance General Purpose - Compute Gen5	West US 2	One Year	SQL Database Managed Instance
SQL Database Single/Elastic Pool Business Critical - Compute Gen4	West US 2	One Year	SQL Database Single/Elastic Pool
SQL Database Single/Elastic Pool Business Critical - Compute Gen5	West US 2	One Year	SQL Database Single/Elastic Pool
SQL Database Single/Elastic Pool General Purpose - Compute Gen4	West US 2	One Year	SQL Database Single/Elastic Pool
SQL Database Single/Elastic Pool General Purpose - Compute Gen5	West US 2	One Year	SQL Database Single/Elastic Pool

Select **Cancel**

Price per unit: <UnitPrice>
34% Estimated savings

The following table describes required fields.

FIELD	DESCRIPTION
Subscription	<p>The subscription used to pay for the SQL Database reserved capacity reservation. The payment method on the subscription is charged the upfront costs for the SQL Database reserved capacity reservation. The subscription type must be an enterprise agreement (offer numbers: MS-AZR-0017P or MS-AZR-0148P) or an individual agreement with pay-as-you-go pricing (offer numbers: MS-AZR-0003P or MS-AZR-0023P). For an enterprise subscription, the charges are deducted from the enrollment's monetary commitment balance or charged as overage. For an individual subscription with pay-as-you-go pricing, the charges are billed to the credit card or invoice payment method on the subscription.</p>
Scope	<p>The vCore reservation's scope can cover one subscription or multiple subscriptions (shared scope). If you select:</p> <p>Shared, the vCore reservation discount is applied to SQL Database instances running in any subscriptions within your billing context. For enterprise customers, the shared scope is the enrollment and includes all subscriptions within the enrollment. For Pay-As-You-Go customers, the shared scope is all Pay-As-You-Go subscriptions created by the account administrator.</p> <p>Single subscription, the vCore reservation discount is applied to SQL Database instances in this subscription.</p> <p>Single resource group, the reservation discount is applied to SQL Database instances in the selected subscription and the selected resource group within that subscription.</p>
Region	The Azure region that's covered by the SQL Database reserved capacity reservation.

FIELD	DESCRIPTION
Deployment Type	The SQL resource type that you want to buy the reservation for.
Performance Tier	The service tier for the SQL Database instances.
Term	One year or three years.
Quantity	The amount of compute resources being purchased within the SQL Database reserved capacity reservation. The quantity is a number of vCores in the selected Azure region and Performance tier that are being reserved and will get the billing discount. For example, if you are running or planning to run SQL Database instances with the total compute capacity of Gen5 16 vCores in the East US region, then you would specify quantity as 16 to maximize the benefit for all instances.

1. Review the cost of the SQL Database reserved capacity reservation in the **Costs** section.
2. Select **Purchase**.
3. Select **View this Reservation** to see the status of your purchase.

Cancel, exchange, or refund reservations

You can cancel, exchange, or refund reservations with certain limitations. For more information, see [Self-service exchanges and refunds for Azure Reservations](#).

vCore size flexibility

vCore size flexibility helps you scale up or down within a performance tier and region, without losing the reserved capacity benefit. SQL Database reserved capacity also provides you with the flexibility to temporarily move your hot databases between pools and single databases as part of your normal operations (within the same region and performance tier) without losing the reserved capacity benefit. By keeping an un-applied buffer in your reservation, you can effectively manage the performance spikes without exceeding your budget.

Limitation

You cannot reserve DTU-based (basic, standard, or premium) SQL databases.

Need help? Contact us

If you have questions or need help, [create a support request](#).

Next steps

The vCore reservation discount is applied automatically to the number of SQL Database instances that match the SQL Database reserved capacity reservation scope and attributes. You can update the scope of the SQL Database reserved capacity reservation through [Azure portal](#), PowerShell, CLI or through the API.

To learn how to manage the SQL Database reserved capacity reservation, see [manage SQL Database reserved capacity](#).

To learn more about Azure Reservations, see the following articles:

- [What are Azure Reservations?](#)
- [Manage Azure Reservations](#)
- [Understand Azure Reservations discount](#)
- [Understand reservation usage for your Pay-As-You-Go subscription](#)
- [Understand reservation usage for your Enterprise enrollment](#)
- [Azure Reservations in Partner Center Cloud Solution Provider \(CSP\) program](#)

Service tiers in the DTU-based purchase model

12/31/2019 • 8 minutes to read • [Edit Online](#)

Service tiers in the DTU-based purchase model are differentiated by a range of compute sizes with a fixed amount of included storage, fixed retention period for backups, and fixed price. All service tiers in the DTU-based purchase model provide flexibility of changing compute sizes with minimal **downtime**; however, there is a switch over period where connectivity is lost to the database for a short amount of time, which can be mitigated using retry logic. Single databases and elastic pools are billed hourly based on service tier and compute size.

IMPORTANT

SQL Database managed instance does not support a DTU-based purchasing model. For more information, see [Azure SQL Database Managed Instance](#).

NOTE

For information about vCore-based service tiers, see [vCore-based service tiers](#). For information about differentiating DTU-based service tiers and vCore-based service tiers, see [Azure SQL Database purchasing models](#).

Compare the DTU-based service tiers

Choosing a service tier depends primarily on business continuity, storage, and performance requirements.

	BASIC	STANDARD	PREMIUM
Target workload	Development and production	Development and production	Development and production
Uptime SLA	99.99%	99.99%	99.99%
Maximum backup retention	7 days	35 days	35 days
CPU	Low	Low, Medium, High	Medium, High
IO throughput (approximate)	1-5 IOPS per DTU	1-5 IOPS per DTU	25 IOPS per DTU
IO latency (approximate)	5 ms (read), 10 ms (write)	5 ms (read), 10 ms (write)	2 ms (read/write)
Columnstore indexing	N/A	S3 and above	Supported
In-memory OLTP	N/A	N/A	Supported

IMPORTANT

The Basic, Standard S0, S1 and S2 service tiers provide less than one vCore (CPU). For CPU-intensive workloads, a service tier of S3 or greater is recommended.

Regarding data storage, the Basic, Standard S0, and S1 service tiers are placed on Standard Page Blobs. Standard Page Blobs use hard disk drive (HDD)-based storage media and are best suited for development, testing, and other infrequently accessed workloads that are less sensitive to performance variability.

NOTE

You can get a free Azure SQL database at the Basic service tier in conjunction with an Azure free account to explore Azure. For information, see [Create a managed cloud database with your Azure free account](#).

Single database DTU and storage limits

Compute sizes are expressed in terms of Database Transaction Units (DTUs) for single databases and elastic Database Transaction Units (eDTUs) for elastic pools. For more on DTUs and eDTUs, see [DTU-based purchasing model](#).

	BASIC	STANDARD	PREMIUM
Maximum storage size	2 GB	1 TB	4 TB
Maximum DTUs	5	3000	4000

IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

Elastic pool eDTU, storage, and pooled database limits

	BASIC	STANDARD	PREMIUM
Maximum storage size per database	2 GB	1 TB	1 TB
Maximum storage size per pool	156 GB	4 TB	4 TB
Maximum eDTUs per database	5	3000	4000
Maximum eDTUs per pool	1600	3000	4000
Maximum number of databases per pool	500	500	100

IMPORTANT

More than 1 TB of storage in the Premium tier is currently available in all regions except: China East, China North, Germany Central, Germany Northeast, West Central US, US DoD regions, and US Government Central. In these regions, the storage max in the Premium tier is limited to 1 TB. For more information, see [P11-P15 current limitations](#).

IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [manage file space in Azure SQL Database](#).

DTU Benchmark

Physical characteristics (CPU, memory, IO) associated to each DTU measure are calibrated using a benchmark that simulates real-world database workload.

Correlating benchmark results to real world database performance

It is important to understand that all benchmarks are representative and indicative only. The transaction rates achieved with the benchmark application will not be the same as those that might be achieved with other applications. The benchmark comprises a collection of different transaction types run against a schema containing a range of tables and data types. While the benchmark exercises the same basic operations that are common to all OLTP workloads, it does not represent any specific class of database or application. The goal of the benchmark is to provide a reasonable guide to the relative performance of a database that might be expected when scaling up or down between compute sizes. In reality, databases are of different sizes and complexity, encounter different mixes of workloads, and will respond in different ways. For example, an IO-intensive application may hit IO thresholds sooner, or a CPU-intensive application may hit CPU limits sooner. There is no guarantee that any particular database will scale in the same way as the benchmark under increasing load.

The benchmark and its methodology are described in more detail below.

Benchmark summary

The benchmark measures the performance of a mix of basic database operations that occur most frequently in online transaction processing (OLTP) workloads. Although the benchmark is designed with cloud computing in mind, the database schema, data population, and transactions have been designed to be broadly representative of the basic elements most commonly used in OLTP workloads.

Schema

The schema is designed to have enough variety and complexity to support a broad range of operations. The benchmark runs against a database comprised of six tables. The tables fall into three categories: fixed-size, scaling, and growing. There are two fixed-size tables; three scaling tables; and one growing table. Fixed-size tables have a constant number of rows. Scaling tables have a cardinality that is proportional to database performance, but doesn't change during the benchmark. The growing table is sized like a scaling table on initial load, but then the cardinality changes in the course of running the benchmark as rows are inserted and deleted.

The schema includes a mix of data types, including integer, numeric, character, and date/time. The schema includes primary and secondary keys, but not any foreign keys - that is, there are no referential integrity constraints between tables.

A data generation program generates the data for the initial database. Integer and numeric data is generated with various strategies. In some cases, values are distributed randomly over a range. In other cases, a set of values is randomly permuted to ensure that a specific distribution is maintained. Text fields are generated

from a weighted list of words to produce realistic looking data.

The database is sized based on a “scale factor.” The scale factor (abbreviated as SF) determines the cardinality of the scaling and growing tables. As described below in the section Users and Pacing, the database size, number of users, and maximum performance all scale in proportion to each other.

Transactions

The workload consists of nine transaction types, as shown in the table below. Each transaction is designed to highlight a particular set of system characteristics in the database engine and system hardware, with high contrast from the other transactions. This approach makes it easier to assess the impact of different components to overall performance. For example, the transaction “Read Heavy” produces a significant number of read operations from disk.

TRANSACTION TYPE	DESCRIPTION
Read Lite	SELECT; in-memory; read-only
Read Medium	SELECT; mostly in-memory; read-only
Read Heavy	SELECT; mostly not in-memory; read-only
Update Lite	UPDATE; in-memory; read-write
Update Heavy	UPDATE; mostly not in-memory; read-write
Insert Lite	INSERT; in-memory; read-write
Insert Heavy	INSERT; mostly not in-memory; read-write
Delete	DELETE; mix of in-memory and not in-memory; read-write
CPU Heavy	SELECT; in-memory; relatively heavy CPU load; read-only

Workload mix

Transactions are selected at random from a weighted distribution with the following overall mix. The overall mix has a read/write ratio of approximately 2:1.

TRANSACTION TYPE	% OF MIX
Read Lite	35
Read Medium	20
Read Heavy	5
Update Lite	20
Update Heavy	3
Insert Lite	3
Insert Heavy	2

TRANSACTION TYPE	% OF MIX
Delete	2
CPU Heavy	10

Users and pacing

The benchmark workload is driven from a tool that submits transactions across a set of connections to simulate the behavior of a number of concurrent users. Although all of the connections and transactions are machine generated, for simplicity we refer to these connections as "users." Although each user operates independently of all other users, all users perform the same cycle of steps shown below:

1. Establish a database connection.
2. Repeat until signaled to exit:
 - Select a transaction at random (from a weighted distribution).
 - Perform the selected transaction and measure the response time.
 - Wait for a pacing delay.
3. Close the database connection.
4. Exit.

The pacing delay (in step 2c) is selected at random, but with a distribution that has an average of 1.0 second. Thus each user can, on average, generate at most one transaction per second.

Scaling rules

The number of users is determined by the database size (in scale-factor units). There is one user for every five scale-factor units. Because of the pacing delay, one user can generate at most one transaction per second, on average.

For example, a scale-factor of 500 (SF=500) database will have 100 users and can achieve a maximum rate of 100 TPS. To drive a higher TPS rate requires more users and a larger database.

Measurement duration

A valid benchmark run requires a steady-state measurement duration of at least one hour.

Metrics

The key metrics in the benchmark are throughput and response time.

- Throughput is the essential performance measure in the benchmark. Throughput is reported in transactions per unit-of-time, counting all transaction types.
- Response time is a measure of performance predictability. The response time constraint varies with class of service, with higher classes of service having a more stringent response time requirement, as shown below.

CLASS OF SERVICE	THROUGHPUT MEASURE	RESPONSE TIME REQUIREMENT
Premium	Transactions per second	95th percentile at 0.5 seconds
Standard	Transactions per minute	90th percentile at 1.0 seconds
Basic	Transactions per hour	80th percentile at 2.0 seconds

Next steps

- For details on specific compute sizes and storage size choices available for single databases, see [SQL](#)

[Database DTU-based resource limits for single databases.](#)

- For details on specific compute sizes and storage size choices available for elastic pools, see [SQL Database DTU-based resource limits](#).

Migrate from the DTU-based model to the vCore-based model

11/7/2019 • 2 minutes to read • [Edit Online](#)

Migrate a database

Migrating a database from the DTU-based purchasing model to the vCore-based purchasing model is similar to upgrading or downgrading between the standard and premium service tiers in the DTU-based purchasing model.

Migrate databases that use geo-replication

Migrating from the DTU-based model to the vCore-based purchasing model is similar to upgrading or downgrading the geo-replication relationships between databases in the standard and premium service tiers.

During migration, you don't have to stop geo-replication, but you must follow these sequencing rules:

- When upgrading, you must upgrade the secondary database first, and then upgrade the primary.
- When downgrading, reverse the order: you must downgrade the primary database first, and then downgrade the secondary.

When you're using geo-replication between two elastic pools, we recommend that you designate one pool as the primary and the other as the secondary. In that case, when you're migrating elastic pools you should use the same sequencing guidance. However, if you have elastic pools that contain both primary and secondary databases, treat the pool with the higher utilization as the primary and follow the sequencing rules accordingly.

The following table provides guidance for specific migration scenarios:

CURRENT SERVICE TIER	TARGET SERVICE TIER	MIGRATION TYPE	USER ACTIONS
Standard	General purpose	Lateral	Can migrate in any order, but need to ensure appropriate vCore sizing*
Premium	Business critical	Lateral	Can migrate in any order, but need to ensure appropriate vCore sizing*
Standard	Business critical	Upgrade	Must migrate secondary first
Business critical	Standard	Downgrade	Must migrate primary first
Premium	General purpose	Downgrade	Must migrate primary first
General purpose	Premium	Upgrade	Must migrate secondary first
Business critical	General purpose	Downgrade	Must migrate primary first
General purpose	Business critical	Upgrade	Must migrate secondary first

* Every 100 DTUs in the standard tier require at least 1 vCore, and every 125 DTUs in the premium tier require at

least 1 vCore.

Migrate failover groups

Migration of failover groups with multiple databases requires individual migration of the primary and secondary databases. During that process, the same considerations and sequencing rules apply. After the databases are converted to the vCore-based purchasing model, the failover group will remain in effect with the same policy settings.

Create a geo-replication secondary database

You can create a geo-replication secondary database (a geo-secondary) only by using the same service tier as you used for the primary database. For databases with a high log-generation rate, we recommend creating the geo-secondary with the same compute size as the primary.

If you're creating a geo-secondary in the elastic pool for a single primary database, make sure the `maxVCore` setting for the pool matches the primary database's compute size. If you're creating a geo-secondary for a primary in another elastic pool, we recommend that the pools have the same `maxVCore` settings.

Use database copy to convert a DTU-based database to a vCore-based database

You can copy any database with a DTU-based compute size to a database with a vCore-based compute size without restrictions or special sequencing as long as the target compute size supports the maximum database size of the source database. The database copy creates a snapshot of the data as of the starting time of the copy operation and doesn't synchronize data between the source and the target.

Next steps

- For the specific compute sizes and storage size choices available for single databases, see [SQL Database vCore-based resource limits for single databases](#).
- For the specific compute sizes and storage size choices available for elastic pools, see [SQL Database vCore-based resource limits for elastic pools](#).

Azure SQL Database service tiers

1/31/2020 • 6 minutes to read • [Edit Online](#)

Azure SQL Database is based on SQL Server database engine architecture that's adjusted for the cloud environment to ensure 99.99 percent availability, even if there is an infrastructure failure. Three service tiers are used in Azure SQL Database, each with a different architectural model. These service tiers are:

- [General purpose](#), which is designed for budget-oriented workloads.
- [Hyperscale](#), which is designed for most business workloads, providing highly scalable storage, read scale-out, and fast database restore capabilities.
- [Business critical](#), which is designed for low-latency workloads with high resiliency to failures and fast failovers.

This article discusses differences between the service tiers, storage and backup considerations for the general purpose and business critical service tiers in the vCore-based purchasing model.

Service tier comparison

The following table describes the key differences between service tiers for the latest generation (Gen5). Note that service tier characteristics might be different in Single Database and Managed Instance.

	RESOURCE TYPE	GENERAL PURPOSE	HYPERSCALE	BUSINESS CRITICAL
Best for		Offers budget oriented balanced compute and storage options.	Most business workloads. Auto-scaling storage size up to 100 TB, fluid vertical and horizontal compute scaling, fast database restore.	OLTP applications with high transaction rate and low IO latency. Offers highest resilience to failures and fast failovers using multiple synchronously updated replicas.
Available in resource type:		Single database / elastic pool / managed instance	Single database	Single database / elastic pool / managed instance
Compute size	Single database / elastic pool	1 to 80 vCores	1 to 80 vCores	1 to 80 vCores
	Managed instance	4, 8, 16, 24, 32, 40, 64, 80 vCores	N/A	4, 8, 16, 24, 32, 40, 64, 80 vCores
	Managed instance pools	2, 4, 8, 16, 24, 32, 40, 64, 80 vCores	N/A	N/A
Storage type	All	Premium remote storage (per instance)	De-coupled storage with local SSD cache (per instance)	Super-fast local SSD storage (per instance)
Database size	Single database / elastic pool	5 GB – 4 TB	Up to 100 TB	5 GB – 4 TB

	RESOURCE TYPE	GENERAL PURPOSE	HYPERSCALE	BUSINESS CRITICAL
	Managed instance	32 GB – 8 TB	N/A	32 GB – 4 TB
Storage size	Single database / elastic pool	5 GB – 4 TB	Up to 100 TB	5 GB – 4 TB
	Managed instance	32 GB – 8 TB	N/A	32 GB – 4 TB
TempDB size	Single database / elastic pool	32 GB per vCore	32 GB per vCore	32 GB per vCore
	Managed instance	24 GB per vCore	N/A	Up to 4 TB - limited by storage size
Log write throughput	Single database	1.875 MB/s per vCore (max 30 MB/s)	100 MB/s	6 MB/s per vCore (max 96 MB/s)
	Managed instance	3 MB/s per vCore (max 22 MB/s)	N/A	4 MB/s per vcore (max 48 MB/s)
Availability	All	99.99%	99.95% with one secondary replica, 99.99% with more replicas	99.99% 99.995% with zone redundant single database
Backups	All	RA-GRS, 7-35 days (7 days by default)	RA-GRS, 7 days, constant time point-in-time recovery (PITR)	RA-GRS, 7-35 days (7 days by default)
In-memory OLTP		N/A	N/A	Available
Read-only replicas		0 built-in 0 - 4 using geo-replication	0 - 4 built-in	1 built-in, included in price 0 - 4 using geo-replication
Pricing/billing	Single database	vCore, reserved storage, and backup storage are charged. IOPS is not charged.	vCore for each replica and used storage are charged. IOPS not yet charged.	vCore, reserved storage, and backup storage are charged. IOPS is not charged.
	Managed Instance	vCore, reserved storage, and backup storage is charged. IOPS is not charged	N/A	vCore, reserved storage, and backup storage is charged. IOPS is not charged.
Discount models		Reserved instances Azure Hybrid Benefit (not available on dev/test subscriptions) Enterprise and Pay-As-You-Go Dev/Test subscriptions	Azure Hybrid Benefit (not available on dev/test subscriptions) Enterprise and Pay-As-You-Go Dev/Test subscriptions	Reserved instances Azure Hybrid Benefit (not available on dev/test subscriptions) Enterprise and Pay-As-You-Go Dev/Test subscriptions

For more information, see the detailed differences between the service tiers in [Single database \(vCore\)](#), [Single](#)

database pools (vCore), Single database (DTU), Single database pools (DTU), and Managed Instance pages.

NOTE

For information about the hyperscale service tier in the vCore-based purchasing model, see [hyperscale service tier](#). For a comparison of the vCore-based purchasing model with the DTU-based purchasing model, see [Azure SQL Database purchasing models and resources](#).

Data and log storage

The following factors affect the amount of storage used for data and log files, and applies to General Purpose and Business Critical. For details on data and log storage in Hyperscale, see [Hyperscale service tier](#).

- The allocated storage is used by data files (MDF) and log files (LDF).
- Each single database compute size supports a maximum database size, with a default maximum size of 32 GB.
- When you configure the required single database size (the size of the MDF file), 30 percent more additional storage is automatically added to support LDF files.
- The storage size for a SQL Database managed instance must be specified in multiples of 32 GB.
- You can select any single database size between 10 GB and the supported maximum.
 - For storage in the standard or general purpose service tiers, increase or decrease the size in 10-GB increments.
 - For storage in the premium or business critical service tiers, increase or decrease the size in 250-GB increments.
- In the general purpose service tier, `tempdb` uses an attached SSD, and this storage cost is included in the vCore price.
- In the business critical service tier, `tempdb` shares the attached SSD with the MDF and LDF files, and the `tempdb` storage cost is included in the vCore price.

IMPORTANT

You are charged for the total storage allocated for MDF and LDF files.

To monitor the current total size of your MDF and LDF files, use [sp_spaceused](#). To monitor the current size of the individual MDF and LDF files, use [sys.database_files](#).

IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

Backups and storage

Storage for database backups is allocated to support the point-in-time restore (PITR) and [long-term retention \(LTR\)](#) capabilities of SQL Database. This storage is allocated separately for each database and billed as two separate per-database charges.

- **PITR:** Individual database backups are copied to [read-access geo-redundant \(RA-GRS\) storage](#) automatically. The storage size increases dynamically as new backups are created. The storage is used by weekly full backups, daily differential backups, and transaction log backups, which are copied every 5 minutes. The storage consumption depends on the rate of change of the database and the retention period for backups. You can configure a separate retention period for each database between 7 and 35 days. A minimum storage amount

equal to 100 percent (1x) of the database size is provided at no extra charge. For most databases, this amount is enough to store 7 days of backups.

- **LTR:** SQL Database offers you the option of configuring long-term retention of full backups for up to 10 years. If you set up an LTR policy, these backups are stored in RA-GRS storage automatically, but you can control how often the backups are copied. To meet different compliance requirements, you can select different retention periods for weekly, monthly, and/or yearly backups. The configuration you choose determines how much storage will be used for LTR backups. To estimate the cost of LTR storage, you can use the LTR pricing calculator. For more information, see [SQL Database long-term retention](#).

Next steps

- For details about the specific compute sizes and storage sizes available for a single database in the general purpose and business critical service tiers, see [SQL Database vCore-based resource limits for single databases](#).
- For details about the specific compute sizes and storage sizes available for elastic pools in the general purpose and business critical service tiers, see [SQL Database vCore-based resource limits for elastic pools](#).

General purpose service tier - Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

NOTE

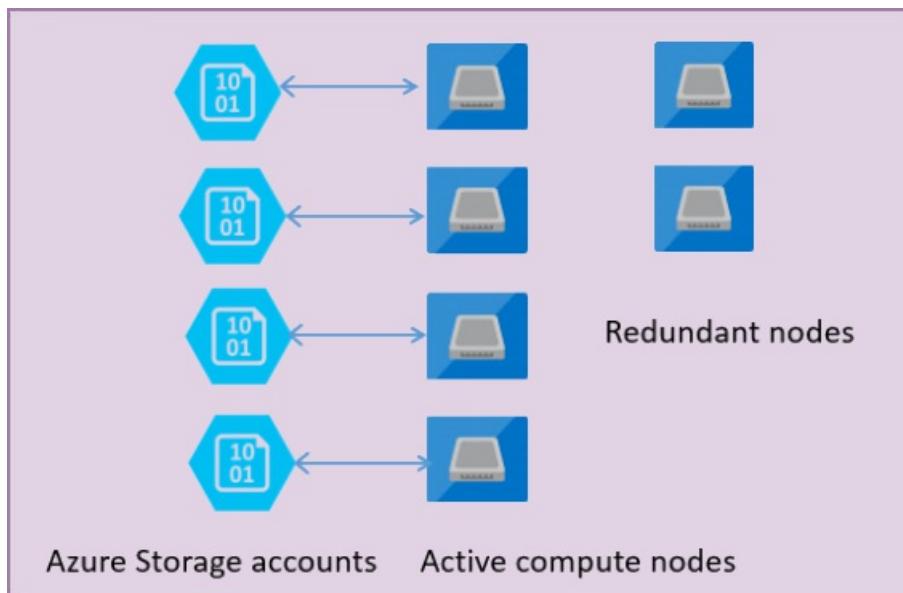
The general-purpose service tier in the vCore-based purchasing model is called the standard service tier in the DTU-based purchasing model. For a comparison of the vCore-based purchasing model with the DTU-based purchasing model, see [Azure SQL Database purchasing models and resources](#).

Azure SQL Database is based on SQL Server database engine architecture adapted for the cloud environment in order to ensure 99.99% availability even in the cases of infrastructure failures. There are three service tiers that are used in Azure SQL Database, each with different architectural models. These service tiers are:

- General purpose
- Business critical
- Hyperscale

The architectural model for the general-purpose service tier is based on a separation of compute and storage. This architectural model relies on high availability and reliability of Azure Blob storage that transparently replicates database files and guarantees no data loss if underlying infrastructure failure happens.

The following figure shows four nodes in standard architectural model with the separated compute and storage layers.



In the architectural model for the general-purpose service tier, there are two layers:

- A stateless compute layer that is running the `sqlservr.exe` process and contains only transient and cached data (for example – plan cache, buffer pool, column store pool). This stateless SQL Server node is operated by Azure Service Fabric that initializes process, controls health of the node, and performs failover to another place if necessary.
- A stateful data layer with database files (.mdf/.ldf) that are stored in Azure Blob storage. Azure Blob storage guarantees that there will be no data loss of any record that is placed in any database file. Azure Storage has built-in data availability/redundancy that ensures that every record in log file or page in data file will be preserved even if SQL Server process crashes.

Whenever database engine or operating system is upgraded, some part of underlying infrastructure fails, or if some critical issue is detected in SQL Server process, Azure Service Fabric will move the stateless SQL Server process to another stateless compute node. There is a set of spare nodes that is waiting to run new compute service if a failover of the primary node happens in order to minimize failover time. Data in Azure storage layer is not affected, and data/log files are attached to newly initialized SQL Server process. This process guarantees 99.99% availability, but it might have some performance impacts on heavy workload that is running due to transition time and the fact the new SQL Server node starts with cold cache.

When to choose this service tier

General Purpose service tier is a default service tier in Azure SQL Database that is designed for most of the generic workloads. If you need a fully managed database engine with 99.99% SLA with storage latency between 5 and 10 ms that match Azure SQL IaaS in most of the cases, General Purpose tier is the option for you.

Next steps

- Find resource characteristics (number of cores, IO, memory) of General Purpose/Standard tier in [Managed Instance](#), Single database in [vCore model](#) or [DTU model](#), or Elastic pool in [vCore model](#) and [DTU model](#).
- Learn about [Business Critical](#) and [Hyperscale](#) tiers.
- Learn about [Service Fabric](#).
- For more options for high availability and disaster recovery, see [Business Continuity](#).

Business Critical tier - Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

NOTE

Business Critical tier is called Premium in DTU purchasing model. For a comparison of the vCore-based purchasing model with the DTU-based purchasing model, see [Azure SQL Database purchasing models and resources](#).

Azure SQL Database is based on SQL Server Database Engine architecture that is adjusted for the cloud environment in order to ensure 99.99% availability even in the cases of infrastructure failures. There are three architectural models that are used in Azure SQL Database:

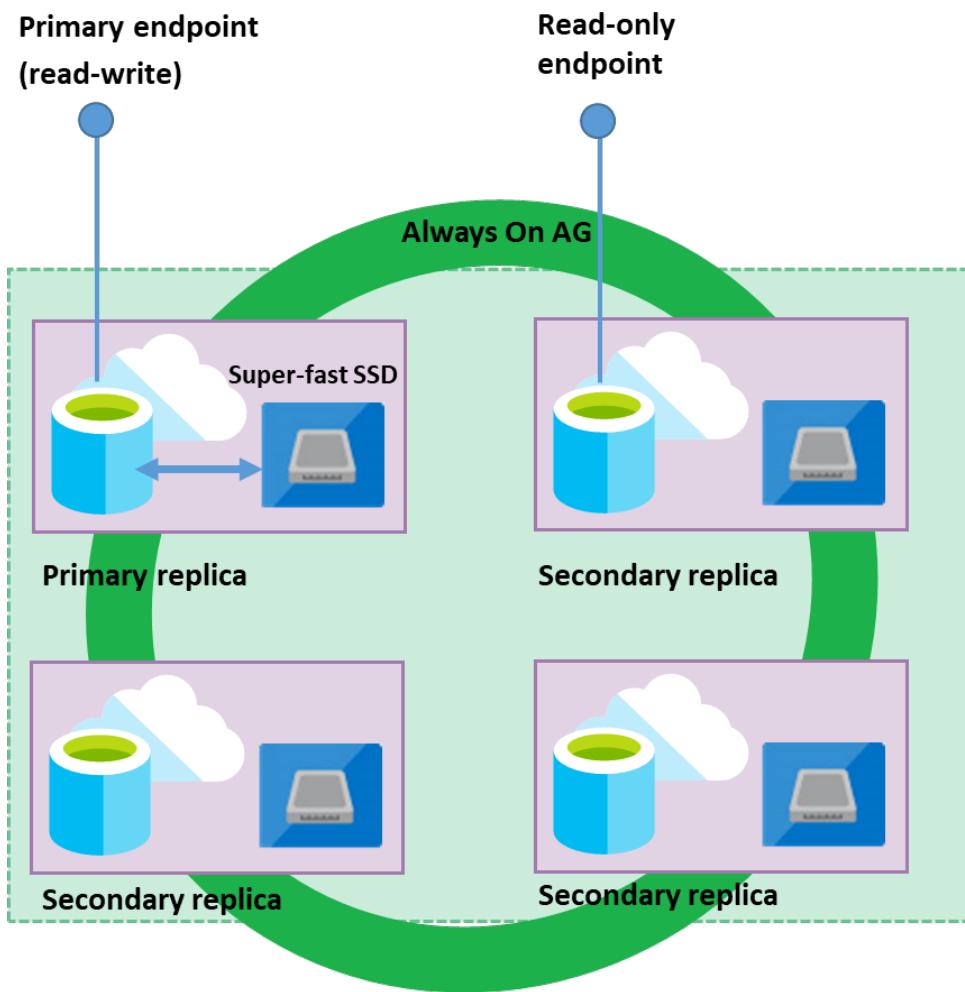
- General Purpose/Standard
- Business Critical/Premium
- Hyperscale

Premium/Business Critical service tier model is based on a cluster of database engine processes. This architectural model relies on a fact that there is always a quorum of available database engine nodes and has minimal performance impact on your workload even during maintenance activities.

Azure upgrades and patches underlying operating system, drivers, and SQL Server Database Engine transparently with the minimal down-time for end users.

Premium availability is enabled in Premium and Business Critical service tiers of Azure SQL Database and it is designed for intensive workloads that cannot tolerate any performance impact due to the ongoing maintenance operations.

In the premium model, Azure SQL database integrates compute and storage on the single node. High availability in this architectural model is achieved by replication of compute (SQL Server Database Engine process) and storage (locally attached SSD) deployed in four node cluster, using technology similar to SQL Server [Always On Availability Groups](#).



Business Critical service tier: collocated compute and storage

Both the SQL database engine process and underlying mdf/ldf files are placed on the same node with locally attached SSD storage providing low latency to your workload. High availability is implemented using technology similar to SQL Server [Always On Availability Groups](#). Every database is a cluster of database nodes with one primary database that is accessible for customer workload, and three secondary processes containing copies of data. The primary node constantly pushes the changes to secondary nodes in order to ensure that the data is available on secondary replicas if the primary node crashes for any reason. Failover is handled by the SQL Server Database Engine – one secondary replica becomes the primary node and a new secondary replica is created to ensure enough nodes in the cluster. The workload is automatically redirected to the new primary node.

In addition, Business Critical cluster has built-in [Read Scale-Out](#) capability that provides free-of-charge built-in read-only node that can be used to run read-only queries (for example reports) that should not affect performance of your primary workload.

When to choose this service tier?

Business Critical service tier is designed for the applications that require low-latency responses from the underlying SSD storage (1-2 ms in average), fast recovery if the underlying infrastructure fails, or need to off-load reports, analytics, and read-only queries to the free of charge readable secondary replica of the primary database.

The key reasons why you should choose Business Critical service tier instead of General Purpose tier are:

- Low IO latency requirements – workload that needs the fast response from the storage layer (1-2 milliseconds in average) should use Business Critical tier.
- Frequent communication between application and database. Application that cannot leverage application-layer

caching or [request batching](#) and need to send many SQL queries that must be quickly processed are good candidates for Business Critical tier.

- Large number of updates – insert, update, and delete operations modify the data pages in memory (dirty page) that must be saved to data files with [CHECKPOINT](#) operation. Potential database engine process crash or a failover of the database with a large number of dirty pages might increase recovery time in General Purpose tier. Use Business Critical tier if you have a workload that causes many in-memory changes.
- Long running transactions that modify data. Transactions that are opened for a longer time prevent truncation of log file that might increase log size and number of [Virtual log files \(VLF\)](#). High number of VLF can slow down recovery of database after failover.
- Workload with reporting and analytic queries that can be redirected to the free-of-charge secondary read-only replica.
- Higher resiliency and faster recovery from the failures. In a case of system failure, the database on primary instance will be disabled and one of the secondary replicas will be immediately become new read-write primary database that is ready to process the queries. Database engine doesn't need to analyze and redo transactions from the log file and load all data in the memory buffer.
- Advanced data corruption protection - Business Critical tier leverages database replicas behind-the-scenes for business continuity purposes, and so the service also then leverages automatic page repair, which is the same technology used for SQL Server database [mirroring and availability groups](#). In the event that a replica cannot read a page due to a data integrity issue, a fresh copy of the page will be retrieved from another replica, replacing the unreadable page without data loss or customer downtime. This functionality is applicable in General Purpose tier if the database has geo-secondary replica.
- Higher availability - Business Critical tier in Multi-AZ configuration guarantees 99.995% availability, compared to 99.99% of General Purpose tier.
- Fast geo-recovery - Business Critical tier configured with geo-replication has a guaranteed Recovery point objective (RPO) of 5 sec and Recovery time objective (RTO) of 30 sec for 100% of deployed hours.

Next steps

- Find resource characteristics (number of cores, IO, memory) of Business Critical tier in [Managed Instance](#), Single database in [vCore model](#) or [DTU model](#), or Elastic pool in [vCore model](#) and [DTU model](#).
- Learn about [General Purpose](#) and [Hyperscale](#) tiers.
- Learn about [Service Fabric](#).
- For more options for high availability and disaster recovery, see [Business Continuity](#).

Hyperscale service tier

2/24/2020 • 15 minutes to read • [Edit Online](#)

Azure SQL Database is based on SQL Server Database Engine architecture that is adjusted for the cloud environment in order to ensure 99.99% availability even in the cases of infrastructure failures. There are three architectural models that are used in Azure SQL Database:

- General Purpose/Standard
- Hyperscale
- Business Critical/Premium

The Hyperscale service tier in Azure SQL Database is the newest service tier in the vCore-based purchasing model. This service tier is a highly scalable storage and compute performance tier that leverages the Azure architecture to scale out the storage and compute resources for an Azure SQL Database substantially beyond the limits available for the General Purpose and Business Critical service tiers.

NOTE

For details on the General Purpose and Business Critical service tiers in the vCore-based purchasing model, see [General Purpose](#) and [Business Critical](#) service tiers. For a comparison of the vCore-based purchasing model with the DTU-based purchasing model, see [Azure SQL Database purchasing models and resources](#).

What are the Hyperscale capabilities

The Hyperscale service tier in Azure SQL Database provides the following additional capabilities:

- Support for up to 100 TB of database size
- Nearly instantaneous database backups (based on file snapshots stored in Azure Blob storage) regardless of size with no IO impact on compute resources
- Fast database restores (based on file snapshots) in minutes rather than hours or days (not a size of data operation)
- Higher overall performance due to higher log throughput and faster transaction commit times regardless of data volumes
- Rapid scale out - you can provision one or more read-only nodes for offloading your read workload and for use as hot-standbys
- Rapid Scale up - you can, in constant time, scale up your compute resources to accommodate heavy workloads as and when needed, and then scale the compute resources back down when not needed.

The Hyperscale service tier removes many of the practical limits traditionally seen in cloud databases. Where most other databases are limited by the resources available in a single node, databases in the Hyperscale service tier have no such limits. With its flexible storage architecture, storage grows as needed. In fact, Hyperscale databases aren't created with a defined max size. A Hyperscale database grows as needed - and you are billed only for the capacity you use. For read-intensive workloads, the Hyperscale service tier provides rapid scale-out by provisioning additional read replicas as needed for offloading read workloads.

Additionally, the time required to create database backups or to scale up or down is no longer tied to the volume of data in the database. Hyperscale databases can be backed up virtually instantaneously. You can also scale a database in the tens of terabytes up or down in minutes. This capability frees you from concerns about being boxed in by your initial configuration choices.

For more information about the compute sizes for the Hyperscale service tier, see [Service tier characteristics](#).

Who should consider the Hyperscale service tier

The Hyperscale service tier is intended for most business workloads as it provides great flexibility and high performance with independently scalable compute and storage resources. With the ability to auto-scale storage up to 100 TB, it's a great choice for customers who:

- Have large databases on-premises and want to modernize their applications by moving to the cloud
- Are already in the cloud and are limited by the maximum database size restrictions of other service tiers (1-4 TB)
- Have smaller databases, but require fast vertical and horizontal compute scaling, high performance, instant backup, and fast database restore.

The Hyperscale service tier supports a broad range of SQL Server workloads, from pure OLTP to pure analytics, but it is primarily optimized for OLTP and hybrid transaction and analytical processing (HTAP) workloads.

IMPORTANT

Elastic pools do not support the Hyperscale service tier.

Hyperscale pricing model

Hyperscale service tier is only available in [vCore model](#). To align with the new architecture, the pricing model is slightly different from General Purpose or Business Critical service tiers:

- **Compute:**

The Hyperscale compute unit price is per replica. The [Azure Hybrid Benefit](#) price is applied to read scale replicas automatically. We create a primary replica and one read-only replica per Hyperscale database by default. Users may adjust the total number of replicas including the primary from 1-5.

- **Storage:**

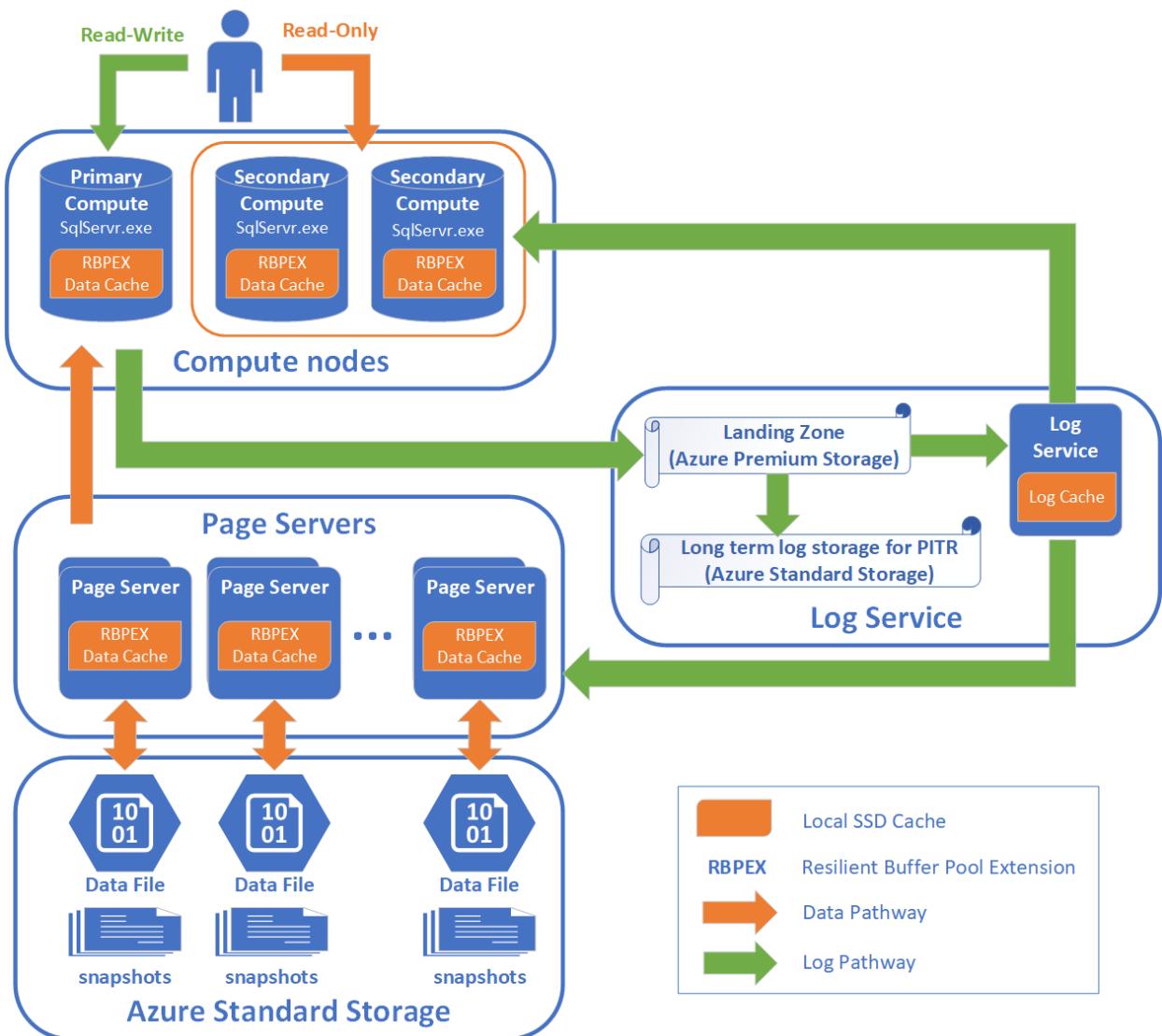
You don't need to specify the max data size when configuring a Hyperscale database. In the hyperscale tier, you are charged for storage for your database based on actual allocation. Storage is automatically allocated between 40 GB and 100 TB, in 10 GB increments 10 GB. Multiple data files can grow at the same time if needed. A Hyperscale database is created with a starting size of 10 GB and it starts growing by 10 GB every 10 minutes, until it reaches the size of 40 GB.

For more information about Hyperscale pricing, see [Azure SQL Database Pricing](#)

Distributed functions architecture

Unlike traditional database engines that have centralized all of the data management functions in one location/process (even so called distributed databases in production today have multiple copies of a monolithic data engine), a Hyperscale database separates the query processing engine, where the semantics of various data engines diverge, from the components that provide long-term storage and durability for the data. In this way, the storage capacity can be smoothly scaled out as far as needed (initial target is 100 TB). Read-only replicas share the same storage components so no data copy is required to spin up a new readable replica.

The following diagram illustrates the different types of nodes in a Hyperscale database:



A Hyperscale database contains the following different types of components:

Compute

The compute node is where the relational engine lives, so all the language elements, query processing, and so on, occur. All user interactions with a Hyperscale database happen through these compute nodes. Compute nodes have SSD-based caches (labeled RBPEX - Resilient Buffer Pool Extension in the preceding diagram) to minimize the number of network round trips required to fetch a page of data. There is one primary compute node where all the read-write workloads and transactions are processed. There are one or more secondary compute nodes that act as hot standby nodes for failover purposes, as well as act as read-only compute nodes for offloading read workloads (if this functionality is desired).

Page server

Page servers are systems representing a scaled-out storage engine. Each page server is responsible for a subset of the pages in the database. Nominally, each page server controls between 128 GB and 1 TB of data. No data is shared on more than one page server (outside of replicas that are kept for redundancy and availability). The job of a page server is to serve database pages out to the compute nodes on demand, and to keep the pages updated as transactions update data. Page servers are kept up-to-date by playing log records from the log service. Page servers also maintain SSD-based caches to enhance performance. Long-term storage of data pages is kept in Azure Storage for additional reliability.

Log service

The log service accepts log records from the primary compute replica, persists them in a durable cache, and forwards the log records to the rest of compute replicas (so they can update their caches) as well as the relevant page server(s), so that the data can be updated there. In this way, all data changes from the primary compute

replica are propagated through the log service to all the secondary compute replicas and page servers. Finally, the log records are pushed out to long-term storage in Azure Storage, which is a virtually infinite storage repository. This mechanism removes the need for frequent log truncation. The log service also has local cache to speed up access to log records.

Azure storage

Azure Storage contains all data files in a database. Page servers keep data files in Azure Storage up to date. This storage is used for backup purposes, as well as for replication between Azure regions. Backups are implemented using storage snapshots of data files. Restore operations using snapshots are fast regardless of data size. Data can be restored to any point in time within the backup retention period of the database.

Backup and restore

Backups are file-snapshot based and hence they are nearly instantaneous. Storage and compute separation enables pushing down the backup/restore operation to the storage layer to reduce the processing burden on the primary compute replica. As a result, database backup does not impact performance of the primary compute node; similarly, restores are done by reverting to file snapshots, and as such are not a size of data operation. Restore is a constant-time operation, and even multiple-terabyte databases can be restored in minutes instead of hours or days. Creation of new databases by restoring an existing backup also takes advantage of this feature: creating database copies for development or testing purposes, even of terabyte sized databases, is doable in minutes.

Scale and performance advantages

With the ability to rapidly spin up/down additional read-only compute nodes, the Hyperscale architecture allows significant read scale capabilities and can also free up the primary compute node for serving more write requests. Also, the compute nodes can be scaled up/down rapidly due to the shared-storage architecture of the Hyperscale architecture.

Create a Hyperscale database

A Hyperscale database can be created using the [Azure portal](#), [T-SQL](#), [PowerShell](#) or [CLI](#). Hyperscale databases are available only using the [vCore-based purchasing model](#).

The following T-SQL command creates a Hyperscale database. You must specify both the edition and service objective in the `CREATE DATABASE` statement. Refer to the [resource limits](#) for a list of valid service objectives.

```
-- Create a Hyperscale Database
CREATE DATABASE [HyperscaleDB1] (EDITION = 'Hyperscale', SERVICE_OBJECTIVE = 'HS_Gen5_4');
GO
```

This will create a Hyperscale database on Gen5 hardware with 4 cores.

Migrate an existing Azure SQL Database to the Hyperscale service tier

You can move your existing Azure SQL databases to Hyperscale using the [Azure portal](#), [T-SQL](#), [PowerShell](#) or [CLI](#). At this time, this is a one-way migration. You can't move databases from Hyperscale to another service tier, other than by exporting and importing data. For proofs of concept (POCs), we recommend making a copy of your production databases, and migrating the copy to Hyperscale. Migrating an existing Azure SQL database to the Hyperscale tier is a size of data operation.

The following T-SQL command moves a database into the Hyperscale service tier. You must specify both the edition and service objective in the `ALTER DATABASE` statement.

```
-- Alter a database to make it a Hyperscale Database  
ALTER DATABASE [DB2] MODIFY (EDITION = 'Hyperscale', SERVICE_OBJECTIVE = 'HS_Gen5_4');  
GO
```

Connect to a read-scale replica of a Hyperscale database

In Hhyperscale databases, the `ApplicationIntent` argument in the connection string provided by the client dictates whether the connection is routed to the write replica or to a read-only secondary replica. If the `ApplicationIntent` set to `READONLY` and the database does not have a secondary replica, connection will be routed to the primary replica and defaults to `ReadWrite` behavior.

```
-- Connection string with application intent  
Server=tcp:<myserver>.database.windows.net;Database=<mydatabase>;ApplicationIntent=ReadOnly;User ID=<myLogin>;Password=<myPassword>;Trusted_Connection=False; Encrypt=True;
```

Hyperscale secondary replicas are all identical, using the same Service Level Objective as the primary replica. If more than one secondary replica is present, the workload is distributed across all available secondaries. Each secondary replica is updated independently, thus different replicas could have different data latency relative to the primary replica.

Database High Availability in Hhyperscale

As in all other service tiers, Hyperscale guarantees data durability for committed transactions regardless of compute replica availability. The extent of downtime due to the primary replica becoming unavailable depends on the type of failover (planned vs. unplanned), and on the presence of at least one secondary replica. In a planned failover (i.e. a maintenance event), the system either creates the new primary replica before initiating a failover, or uses an existing secondary replica as the failover target. In an unplanned failover (i.e. a hardware failure on the primary replica), the system uses a secondary replica as a failover target if one exists, or creates a new primary replica from the pool of available compute capacity. In the latter case, downtime duration is longer due to extra steps required to create the new primary replica.

For Hyperscale SLA, see [SLA for Azure SQL Database](#).

Disaster Recovery for Hhyperscale Databases

Restoring a Hhyperscale database to a different geography

If you need to restore an Azure SQL Database Hyperscale DB to a region other than the one it is currently hosted in, as part of a disaster recovery operation or drill, relocation, or any other reason, the primary method is to do a geo-restore of the database. This involves exactly the same steps as what you would use to restore any other AZURE SQL DB to a different region:

1. Create a SQL Database server in the target region if you do not already have an appropriate server there. This server should be owned by the same subscription as the original (source) server.
2. Follow the instructions in the [geo-restore](#) topic of the page on restoring Azure SQL Databases from automatic backups.

NOTE

Because the source and target are in separate regions, the database cannot share snapshot storage with the source database as in non-geo restores, which complete extremely quickly. In the case of a geo-restore of a Hyperscale database, it will be a size-of-data operation, even if the target is in the paired region of the geo-replicated storage. That means that doing a geo-restore will take time proportional to the size of the database being restored. If the target is in the paired region, the copy will be within a region, which will be significantly faster than a cross-region copy, but it will still be a size-of-data operation.

Available regions

The Azure SQL Database Hyperscale tier is currently available in the following regions:

- Australia East
- Australia Southeast
- Brazil South
- Canada Central
- Central US
- China East 2
- China North 2
- East Asia
- East US
- East Us 2
- France Central
- Japan East
- Japan West
- Korea Central
- Korea South
- North Central US
- North Europe
- South Africa North
- South Central US
- Southeast Asia
- UK South
- UK West
- West Europe
- West US
- West US 2

If you want to create Hyperscale database in a region that is not listed as supported, you can send an onboarding request via Azure portal. For instructions, see [Request quota increases for Azure SQL Database](#) for instructions. When submitting your request, use the following guidelines:

- Use the [Other quota request](#) SQL database quota type.
- In the text details, add the compute SKU/total cores including readable replicas.
- Also specify the estimated TB.

Known limitations

These are the current limitations to the Hyperscale service tier as of GA. We are actively working to remove as

many of these limitations as possible.

ISSUE	DESCRIPTION
The Manage Backups pane for a logical server does not show Hyperscale databases will be filtered from SQL server	Hyperscale has a separate method for managing backups, and as such the Long-Term Retention and Point in Time backup Retention settings do not apply / are invalidated. Accordingly, Hyperscale databases do not appear in the Manage Backup pane.
Point-in-time restore	Once a database is migrated into the Hyperscale service tier, restore to a point-in-time prior to the migration is not supported.
Restore of non-Hyperscale DB to Hyperscale and vice-versa	You cannot restore a Hyperscale database into a non-Hyperscale database, nor can you restore a non-Hyperscale database into a Hyperscale database.
If a database has one or more data files larger than 1 TB, migration fails	In some cases, it may be possible to work around this issue by shrinking the large files to be less than 1 TB. If migrating a database being used during the migration process, make sure that no file gets larger than 1 TB. Use the following query to determine the size of database files. <pre>SELECT *, name AS file_name, size * 8. / 1024 / 1024 AS file_size_GB FROM sys.database_files WHERE type_desc = 'ROWS'</pre> ;
Managed Instance	Azure SQL Database Managed Instance is not currently supported with Hyperscale databases.
Elastic Pools	Elastic Pools are not currently supported with SQL Database Hyperscale.
Migration to Hyperscale is currently a one-way operation	Once a database is migrated to Hyperscale, it cannot be migrated directly to a non-Hyperscale service tier. At present, the only way to migrate a database from Hyperscale to non-Hyperscale is to export/import using a BACPAC file or other data movement technologies (Bulk Copy, Azure Data Factory, Azure Databricks, SSIS, etc.)
Migration of databases with persistent in-memory objects	Hyperscale only supports non persistent In-Memory objects (table types, native SPs and functions). Persistent In-Memory tables and other objects must be dropped and recreated as non-In-Memory objects before migrating a database to the Hyperscale service tier.
Change Tracking	Change Tracking is currently in public preview and can be enabled on new or existing Hyperscale databases.
Geo Replication	You cannot yet configure geo-replication for Azure SQL Database Hyperscale.
Database Copy	You cannot yet use Database Copy to create a new database in Azure SQL Hyperscale.

ISSUE	DESCRIPTION
TDE/AKV Integration	Transparent Database Encryption using Azure Key Vault (commonly referred to as Bring-Your-Own-Key or BYOK) is not yet supported for Azure SQL Database Hyperscale, however TDE with Service Managed Keys is fully supported.
Intelligent Database Features	With the exception of the "Force Plan" option, all other Automatic tuning options are not yet supported on Hyperscale: options may appear to be enabled, but there won't be any recommendations or actions made.
Query Performance Insights	Query Performance Insights is currently not supported for Hyperscale databases.
Shrink Database	DBCC SHRINKDATABASE or DBCC SHRINKFILE is not currently supported for Hyperscale databases.
Database integrity check	DBCC CHECKDB is not currently supported for Hyperscale databases. See Data Integrity in Azure SQL Database for details on data integrity management in Azure SQL Database.

Next steps

- For an FAQ on Hyperscale, see [Frequently asked questions about Hyperscale](#).
- For information about service tiers, see [Service tiers](#)
- See [Overview of resource limits on a logical server](#) for information about limits at the server and subscription levels.
- For purchasing model limits for a single database, see [Azure SQL Database vCore-based purchasing model limits for a single database](#).
- For a features and comparison list, see [SQL common features](#).

Azure SQL Database Hyperscale FAQ

1/2/2020 • 19 minutes to read • [Edit Online](#)

This article provides answers to frequently asked questions for customers considering a database in the Azure SQL Database Hhyperscale service tier, referred to as just Hhyperscale in the remainder of this FAQ. This article describes the scenarios that Hhyperscale supports and the features that are compatible with Hhyperscale.

- This FAQ is intended for readers who have a brief understanding of the Hhyperscale service tier and are looking to have their specific questions and concerns answered.
- This FAQ isn't meant to be a guidebook or answer questions on how to use a Hhyperscale database. For an introduction to Hhyperscale, we recommend you refer to the [Azure SQL Database Hhyperscale](#) documentation.

General Questions

What is a Hhyperscale database

A Hhyperscale database is an Azure SQL database in the Hhyperscale service tier that is backed by the Hhyperscale scale-out storage technology. A Hhyperscale database supports up to 100 TB of data and provides high throughput and performance, as well as rapid scaling to adapt to the workload requirements. Scaling is transparent to the application – connectivity, query processing, etc. work like any other Azure SQL database.

What resource types and purchasing models support Hhyperscale

The Hhyperscale service tier is only available for single databases using the vCore-based purchasing model in Azure SQL Database.

How does the Hhyperscale service tier differ from the General Purpose and Business Critical service tiers

The vCore-based service tiers are differentiated based on database availability and storage type, performance, and maximum size, as described in the following table.

	RESOURCE TYPE	GENERAL PURPOSE	HYPERSCALE	BUSINESS CRITICAL
Best for	All	Offers budget oriented balanced compute and storage options.	Most business workloads. Autoscaling storage size up to 100 TB, fast vertical and horizontal compute scaling, fast database restore.	OLTP applications with high transaction rate and low IO latency. Offers highest resilience to failures and fast failovers using multiple synchronously updated replicas.
Resource type		Single database / elastic pool / managed instance	Single database	Single database / elastic pool / managed instance
Compute size	Single database / elastic pool *	1 to 80 vCores	1 to 80 vCores*	1 to 80 vCores
	Managed instance	8, 16, 24, 32, 40, 64, 80 vCores	N/A	8, 16, 24, 32, 40, 64, 80 vCores

	RESOURCE TYPE	GENERAL PURPOSE	HYPERSCALE	BUSINESS CRITICAL
Storage type	All	Premium remote storage (per instance)	De-coupled storage with local SSD cache (per instance)	Super-fast local SSD storage (per instance)
Storage size	Single database / elastic pool *	5 GB – 4 TB	Up to 100 TB	5 GB – 4 TB
	Managed instance	32 GB – 8 TB	N/A	32 GB – 4 TB
IOPS	Single database	500 IOPS per vCore with 7000 maximum IOPS	Hyperscale is a multi-tiered architecture with caching at multiple levels. Effective IOPS will depend on the workload.	5000 IOPS with 200,000 maximum IOPS
	Managed instance	Depends on file size	N/A	1375 IOPS/vCore
Availability	All	1 replica, no Read Scale-out, no local cache	Multiple replicas, up to 4 Read Scale-out, partial local cache	3 replicas, 1 Read Scale-out, zone-redundant HA, full local storage
Backups	All	RA-GRS, 7-35 day retention (7 days by default)	RA-GRS, 7 day retention, constant time point-in-time recovery (PITR)	RA-GRS, 7-35 day retention (7 days by default)

* Elastic pools are not supported in the Hyperscale service tier

Who should use the Hyperscale service tier

The Hyperscale service tier is intended for customers who have large on-premises SQL Server databases and want to modernize their applications by moving to the cloud, or for customers who are already using Azure SQL Database and want to significantly expand the potential for database growth. Hyperscale is also intended for customers who seek both high performance and high scalability. With Hyperscale, you get:

- Database size up to 100 TB
- Fast database backups regardless of database size (backups are based on storage snapshots)
- Fast database restores regardless of database size (restores are from storage snapshots)
- Higher log throughput regardless of database size and the number of vCores
- Read Scale-out using one or more read-only replicas, used for read offloading and as hot standbys.
- Rapid scale up of compute, in constant time, to be more powerful to accommodate the heavy workload and then scale down, in constant time. This is similar to scaling up and down between a P6 and a P11, for example, but much faster as this is not a size of data operation.

What regions currently support Hyperscale

The Hyperscale service tier is currently available in the regions listed under [Azure SQL Database Hyperscale Overview](#).

Can I create multiple Hyperscale databases per logical server

Yes. For more information and limits on the number of Hyperscale databases per logical server, see [SQL Database resource limits for single and pooled databases on a logical server](#).

What are the performance characteristics of a Hyperscale database

The Hyperscale architecture provides high performance and throughput while supporting large database sizes.

What is the scalability of a Hyperscale database

Hyperscale provides rapid scalability based on your workload demand.

- **Scaling Up/Down**

With Hhyperscale, you can scale up the primary compute size in terms of resources like CPU and memory, and then scale down, in constant time. Because the storage is shared, scaling up and scaling down is not a size of data operation.

- **Scaling In/Out**

With Hhyperscale, you also get the ability to provision one or more additional compute replicas that you can use to serve your read requests. This means that you can use these additional compute replicas as read-only replicas to offload your read workload from the primary compute. In addition to read-only, these replicas also serve as hot-standbys in case of a failover from the primary.

Provisioning of each of these additional compute replicas can be done in constant time and is an online operation. You can connect to these additional read-only compute replicas by setting the `ApplicationIntent` argument on your connection string to `ReadOnly`. Any connections with the `ReadOnly` application intent are automatically routed to one of the additional read-only compute replicas.

Deep Dive Questions

Can I mix Hhyperscale and single databases in a single logical server

Yes, you can.

Does Hhyperscale require my application programming model to change

No, your application programming model stays as is. You use your connection string as usual and the other regular ways to interact with your Hhyperscale database.

What transaction isolation level is the default in a Hhyperscale database

On the primary replica, the default transaction isolation level is RCSI (Read Committed Snapshot Isolation). On the Read Scale-out secondary replicas, the default isolation level is Snapshot.

Can I bring my on-premises or IaaS SQL Server license to Hhyperscale

Yes, [Azure Hybrid Benefit](#) is available for Hhyperscale. Every SQL Server Standard core can map to 1 Hhyperscale vCores. Every SQL Server Enterprise core can map to 4 Hhyperscale vCores. You don't need a SQL license for secondary replicas. The Azure Hybrid Benefit price will be automatically applied to Read Scale-out (secondary) replicas.

What kind of workloads is Hhyperscale designed for

Hhyperscale supports all SQL Server workloads, but it is primarily optimized for OLTP. You can bring Hybrid (HTAP) and Analytical (data mart) workloads as well.

How can I choose between Azure SQL Data Warehouse and Azure SQL Database Hhyperscale

If you are currently running interactive analytics queries using SQL Server as a data warehouse, Hhyperscale is a great option because you can host small and mid-size data warehouses (such as a few TB up to 100 TB) at a lower cost, and you can migrate your SQL Server data warehouse workloads to Hhyperscale with minimal T-SQL code changes.

If you are running data analytics on a large scale with complex queries and sustained ingestion rates higher than 100 MB/s, or using Parallel Data Warehouse (PDW), Teradata, or other Massively Parallel Processing (MPP) data warehouses, SQL Data Warehouse may be the best choice.

Hyperscale Compute Questions

Can I pause my compute at any time

Not at this time, however you can scale your compute and number of replicas down to reduce cost during non-peak times.

Can I provision a compute replica with extra RAM for my memory-intensive workload

No. To get more RAM, you need to upgrade to a higher compute size. For more information, see [Hyperscale storage and compute sizes](#).

Can I provision multiple compute replicas of different sizes

No.

How many Read Scale-out replicas are supported

The Hyperscale databases are created with one Read Scale-out replica (two replicas including primary) by default. You can scale the number of read-only replicas between 0 and 4 using [Azure portal](#) or [REST API](#).

For high availability, do I need to provision additional compute replicas

In Hyperscale databases, data resiliency is provided at the storage level. You only need one replica to provide resiliency. When the compute replica is down, a new replica is created automatically with no data loss.

However, if there's only one replica, it may take some time to build the local cache in the new replica after failover. During the cache rebuild phase, the database fetches data directly from the page servers, resulting in higher storage latency and degraded query performance.

For mission-critical apps that require high availability with minimal failover impact, you should provision at least 2 compute replicas including the primary compute replica. This is the default configuration. That way there is a hot-standby replica available that serves as a failover target.

Data Size and Storage Questions

What is the maximum database size supported with Hyperscale

100 TB.

What is the size of the transaction log with Hyperscale

The transaction log with Hyperscale is practically infinite. You do not need to worry about running out of log space on a system that has a high log throughput. However, log generation rate might be throttled for continuous aggressively writing workloads. The peak sustained log generation rate is 100 MB/s.

Does my `tempdb` scale as my database grows

Your `tempdb` database is located on local SSD storage and is sized proportionally to the compute size that you provision. Your `tempdb` is optimized to provide maximum performance benefits. `tempdb` size is not configurable and is managed for you.

Does my database size automatically grow, or do I have to manage the size of data files

Your database size automatically grows as you insert/ingest more data.

What is the smallest database size that Hyperscale supports or starts with

40 GB. A Hyperscale database is created with a starting size of 10 GB. Then, it starts growing by 10 GB every 10 minutes, until it reaches the size of 40 GB. Each of these 10 GB chunks is allocated in a different page server in order to provide more IOPS and higher I/O parallelism. Because of this optimization, even if you choose initial database size smaller than 40 GB, the database will grow to at least 40 GB automatically.

In what increments does my database size grow

Each data file grows by 10 GB. Multiple data files may grow at the same time.

Is the storage in Hyperscale local or remote

In Hyperscale, data files are stored in Azure standard storage. Data is fully cached on local SSD storage, on page servers that are close to the compute replicas. In addition, compute replicas have data caches on local SSD and in memory, to reduce the frequency of fetching data from remote page servers.

Can I manage or define files or filegroups with Hyperscale

No. Data files are added automatically. The common reasons for creating additional filegroups do not apply in the Hyperscale storage architecture.

Can I provision a hard cap on the data growth for my database

No.

How are data files laid out with Hyperscale

The data files are controlled by page servers, with one page server per data file. As the data size grows, data files and associated page servers are added.

Is database shrink supported

No.

Is data compression supported

Yes, including row, page, and columnstore compression.

If I have a huge table, does my table data get spread out across multiple data files

Yes. The data pages associated with a given table can end up in multiple data files, which are all part of the same filegroup. SQL Server uses [proportional fill strategy](#) to distribute data over data files.

Data Migration Questions

Can I move my existing Azure SQL databases to the Hyperscale service tier

Yes. You can move your existing Azure SQL databases to Hyperscale. This is a one-way migration. You can't move databases from Hyperscale to another service tier. For proofs of concept (POCs), we recommend you make a copy of your database and migrate the copy to Hyperscale.

Can I move my Hyperscale databases to other service tiers

No. At this time, you can't move a Hyperscale database to another service tier.

Do I lose any functionality or capabilities after migration to the Hyperscale service tier

Yes. Some of Azure SQL Database features are not supported in Hyperscale yet, including but not limited to long term backup retention. After you migrate your databases to Hyperscale, those features stop working. We expect these limitations to be temporary.

Can I move my on-premises SQL Server database, or my SQL Server database in a cloud virtual machine to Hyperscale

Yes. You can use all existing migration technologies to migrate to Hyperscale, including transactional replication, and any other data movement technologies (Bulk Copy, Azure Data Factory, Azure Databricks, SSIS). See also the [Azure Database Migration Service](#), which supports many migration scenarios.

What is my downtime during migration from an on-premises or virtual machine environment to Hyperscale, and how can I minimize it

Downtime for migration to Hyperscale is the same as the downtime when you migrate your databases to other Azure SQL Database service tiers. You can use [transactional replication](#) to minimize downtime migration for databases up to few TB in size. For very large databases (10+ TB), you can consider to migrate data using ADF, Spark, or other data movement technologies.

How much time would it take to bring in X amount of data to Hyperscale

Hyperscale is capable of consuming 100 MB/s of new/changed data, but the time needed to move data into Azure SQL databases is also affected by available network throughput, source read speed and the target database service level objective.

Can I read data from blob storage and do fast load (like Polybase in SQL Data Warehouse)

You can have a client application read data from Azure Storage and load data into a Hyperscale database (just like you can with any other Azure SQL database). Polybase is currently not supported in Azure SQL Database. As an alternative to provide fast load, you can use [Azure Data Factory](#), or use a Spark job in [Azure Databricks](#) with the [Spark connector for SQL](#). The Spark connector to SQL supports bulk insert.

It is also possible to bulk read data from Azure Blob store using BULK INSERT or OPENROWSET: [Examples of Bulk Access to Data in Azure Blob Storage](#).

Simple recovery or bulk logging model is not supported in Hyperscale. Full recovery model is required to provide high availability and point-in-time recovery. However, Hyperscale log architecture provides better data ingest rate compared to other Azure SQL Database service tiers.

Does Hyperscale allow provisioning multiple nodes for parallel ingesting of large amounts of data

No. Hyperscale is a symmetric multi-processing (SMP) architecture and is not a massively parallel processing (MPP) or a multi-master architecture. You can only create multiple replicas to scale out read-only workloads.

What is the oldest SQL Server version supported for migration to Hyperscale

SQL Server 2005. For more information, see [Migrate to a single database or a pooled database](#). For compatibility issues, see [Resolving database migration compatibility issues](#).

Does Hyperscale support migration from other data sources such as Amazon Aurora, MySQL, PostgreSQL, Oracle, DB2, and other database platforms

Yes. [Azure Database Migration Service](#) supports many migration scenarios.

Business Continuity and Disaster Recovery Questions

What SLAs are provided for a Hyperscale database

See [SLA for Azure SQL Database](#). Additional secondary compute replicas increase availability, up to 99.99% for a database with two or more secondary compute replicas.

Are the database backups managed for me by the Azure SQL Database service

Yes.

How often are the database backups taken

There are no traditional full, differential, and log backups for Hyperscale databases. Instead, there are regular storage snapshots of data files. Log that is generated is simply retained as-is for the configured retention period, allowing restore to any point in time within the retention period.

Does Hyperscale support point in time restore

Yes.

What is the Recovery Point Objective (RPO)/Recovery Time Objective (RTO) for database restore in Hyperscale

The RPO is 0 min. The RTO goal is less than 10 minutes, regardless of database size.

Does database backup affect compute performance on my primary or secondary replicas

No. Backups are managed by the storage subsystem, and leverage storage snapshots. They do not impact user workloads.

Can I perform geo-restore with a Hyperscale database

Yes. Geo-restore is fully supported. Unlike point-in-time restore, geo-restore may require a long running size-of-data operation.

Can I set up geo-replication with Hyperscale database

Not at this time.

Can I take a Hyperscale database backup and restore it to my on-premises server, or on SQL Server in a VM

No. The storage format for Hyperscale databases is different from any released version of SQL Server, and you don't control backups or have access to them. To take your data out of a Hyperscale database, you can extract data using any data movement technologies, i.e. Azure Data Factory, Azure Databricks, SSIS, etc.

Cross-Feature Questions

Do I lose any functionality or capabilities after migration to the Hyperscale service tier

Yes. Some of Azure SQL Database features are not supported in Hyperscale, including but not limited to long term backup retention. After you migrate your databases to Hyperscale, those features stop working.

Will Polybase work with Hyperscale

No. Polybase is not supported in Azure SQL Database.

Does Hyperscale have support for R and Python

Not at this time.

Are compute nodes containerized

No. Hyperscale processes run on a [Service Fabric](#) nodes (VMs), not in containers.

Performance Questions

How much write throughput can I push in a Hyperscale database

Transaction log throughput cap is set to 100 MB/s for any Hyperscale compute size. The ability to achieve this rate depends on multiple factors, including but not limited to workload type, client configuration, and having sufficient compute capacity on the primary compute replica to produce log at this rate.

How many IOPS do I get on the largest compute

IOPS and IO latency will vary depending on the workload patterns. If the data being accessed is cached on the compute replica, you will see similar IO performance as with local SSD.

Does my throughput get affected by backups

No. Compute is decoupled from the storage layer. This eliminates performance impact of backup.

Does my throughput get affected as I provision additional compute replicas

Because the storage is shared and there is no direct physical replication happening between primary and secondary compute replicas, the throughput on primary replica will not be directly affected by adding secondary replicas. However, we may throttle continuous aggressively writing workload on the primary to allow log apply on secondary replicas and page servers to catch up, to avoid poor read performance on secondary replicas.

How do I diagnose and troubleshoot performance problems in a Hyperscale database

For most performance problems, particularly the ones not rooted in storage performance, common SQL Server diagnostic and troubleshooting steps apply. For Hyperscale-specific storage diagnostics, see [SQL Hyperscale performance troubleshooting diagnostics](#).

Scalability Questions

How long would it take to scale up and down a compute replica

Scaling compute up or down should take 5-10 minutes regardless of data size.

Is my database offline while the scaling up/down operation is in progress

No. The scaling up and down will be online.

Should I expect connection drop when the scaling operations are in progress

Scaling up or down results in existing connections being dropped when a failover happens at the end of the scaling operation. Adding secondary replicas does not result in connection drops.

Is the scaling up and down of compute replicas automatic or end-user triggered operation

End-user. Not automatic.

Does the size of my `tempdb` database also grow as the compute is scaled up

Yes. The `tempdb` database will scale up automatically as the compute grows.

Can I provision multiple primary compute replicas, such as a multi-master system, where multiple primary compute heads can drive a higher level of concurrency

No. Only the primary compute replica accepts read/write requests. Secondary compute replicas only accept read-only requests.

Read Scale-out Questions

How many secondary compute replicas can I provision

We create one secondary replica for Hyperscale databases by default. If you want to adjust the number of replicas, you can do so using [Azure portal](#) or [REST API](#).

How do I connect to these secondary compute replicas

You can connect to these additional read-only compute replicas by setting the `ApplicationIntent` argument on your connection string to `ReadOnly`. Any connections marked with `ReadOnly` are automatically routed to one of the additional read-only compute replicas.

How do I validate if I have successfully connected to secondary compute replica using SSMS or other client tools?

You can execute the following T-SQL query: `SELECT DATABASEPROPERTYEX ('<database_name>', 'Updateability')`. The result is `READ_ONLY` if you are connected to a read-only secondary replica, and `READ_WRITE` if you are connected to the primary replica. Note that the database context must be set to the name of the Hyperscale database, not to the `master` database.

Can I create a dedicated endpoint for a Read Scale-out replica

No. You can only connect to Read Scale-out replicas by specifying `ApplicationIntent=ReadOnly`.

Does the system do intelligent load balancing of the read workload

No. A new connection with read-only intent is redirected to an arbitrary Read Scale-out replica.

Can I scale up/down the secondary compute replicas independently of the primary replica

No. The secondary compute replica are also used as high availability failover targets, so they need to have the same configuration as the primary to provide expected performance after failover.

Do I get different `tempdb` sizing for my primary compute and my additional secondary compute replicas

No. Your `tempdb` database is configured based on the compute size provisioning, your secondary compute replicas are the same size as the primary compute.

Can I add indexes and views on my secondary compute replicas

No. Hyperscale databases have shared storage, meaning that all compute replicas see the same tables, indexes, and

views. If you want additional indexes optimized for reads on secondary, you must add them on the primary.

How much delay is there going to be between the primary and secondary compute replicas

Data latency from the time a transaction is committed on the primary to the time it is visible on a secondary depends on current log generation rate. Typical data latency is in low milliseconds.

Next Steps

For more information about the Hyperscale service tier, see [Hyperscale service tier](#).

Azure SQL Database Premium RS service tier (preview) is being retired - options for migration

11/7/2019 • 5 minutes to read • [Edit Online](#)

In February 2018, Microsoft announced that the Premium RS service tier in Azure SQL Database would not be released for general availability and would no longer be supported after January 31, 2019. This end of support deadline has been extended to June 30, 2019. This article explains your options for migrating from the Premium RS service tier to another service tier. After June 30, 2019, Microsoft will automatically migrate your Premium RS databases to a generally available service tier that most closely matches the performance requirements of your Premium RS database.

The following are the migration destinations and pricing options that may be suitable for Premium RS customers:

- vCore service tiers

The **General Purpose** and **Business Critical** service tiers in the [vCore-based purchase model](#). These two service tiers are in general availability. The vCore-based purchasing model also offers the **Hyperscale** service tier that adapts on-demand to your workload's needs with auto-scaling up to 100 TB per database. The Hyperscale service tier provides IO performance comparable to the Premium service tier in the [DTU-based purchasing model](#) at a price closer to the Premium RS service tier.

- Dev/Test pricing

[Dev/test pricing](#) provides savings up to 55% versus license-included rates with your Visual Studio subscription.

- Azure Hybrid Benefit and reserved capacity pricing

[Azure Hybrid Benefit and reserved capacity pricing](#) provide savings up to 80% versus license-included rates. For more information on these options, see [Azure Hybrid Benefit for SQL Server](#) and [Azure SQL Database reserved capacity](#).

Act now to migrate your Premium RS databases to alternative SQL Database service tiers

Review the guidance in this article along with our pricing and documentation to determine the right migration destination(s) for your Premium RS workloads.

Migrate compute-intensive workloads and save

For your compute-intensive Premium RS workloads, we recommend migrating to our generally available vCore-based General Purpose service tier and save more versus license-included rates using the Azure Hybrid Benefit for SQL Server and reserved capacity offers. If you would rather remain on a DTU-based purchasing option, you can migrate your compute-intensive Premium RS databases to a Standard service tier and still save versus the Premium RS general availability pricing (if it had gone into general availability).

WARNING

Migrating your Premium RS workloads to DTU-based Premium service tiers may increase monthly costs versus current Premium RS pricing. We recommend considering the Hyperscale or Business Critical tiers with Azure Hybrid Benefit and reserved capacity pricing to maintain similar or lower costs than Premium RS.

Premium RS databases

IF YOU ARE CURRENTLY ON...	MIGRATE TO COMPARABLE VCORE-BASED...	MIGRATE TO COMPARABLE DTU-BASED...
Premium RS 1	General Purpose 1 vCore (Gen4)	Standard 3
Premium RS 2	General Purpose 2 vCores (Gen4)	Standard 4
Premium RS 4	General Purpose 4 vCores (Gen4)	Standard 6
Premium RS 6	General Purpose 6 vCores (Gen4)	Standard 7

Premium RS pools

IF YOU ARE CURRENTLY ON...	MIGRATE TO COMPARABLE VCORE-BASED...	MIGRATE TO COMPARABLE DTU-BASED...
Premium RS pool 125 DTU	General Purpose 1 vCore (Gen4)	Standard pool 100 eDTUs
Premium RS pool 250 DTU	General Purpose 2 vCores (Gen4)	Standard pool 250 eDTUs
Premium RS pool 500 DTU	General Purpose 4 vCores (Gen4)	Standard pool 500 eDTUs
Premium RS pool 1000 DTU	General Purpose 8 vCores (Gen4)	Standard pool 1000 eDTUs

Optimize savings and performance for your IO-intensive workloads

We recommend migrating your IO-intensive single databases to our vCore-based Hyperscale tier, currently in preview, and your IO-intensive database pools to our generally available Business Critical tier, for the optimal combination of performance and cost. The following vCore-based options will maintain or improve your current performance and may save you money when combined with the Azure Hybrid Benefit and reserved capacity pricing.

IF YOU ARE CURRENTLY ON...	MIGRATE TO COMPARABLE VCORE-BASED...	MIGRATE TO COMPARABLE DTU-BASED...
Premium RS 1	Hyperscale 1 vCore (Gen4) or Business Critical 1 vCore (Gen4)	Premium 1
Premium RS 2	Hyperscale 2 vCores (Gen4) or Business Critical 2 vCores (Gen4)	Premium 2
Premium RS 4	Hyperscale 4 vCores (Gen4) or Business Critical 4 vCores (Gen4)	Premium 4
Premium RS 6	Hyperscale 6 vCores (Gen4) or Business Critical 6 vCores (Gen4)	Premium 6

IF YOU ARE CURRENTLY ON...	MIGRATE TO COMPARABLE VCORE-BASED...	MIGRATE TO COMPARABLE DTU-BASED...
Premium RS pool 125 DTU	Business Critical 2 vCores (Gen4)	Premium pool 125 eDTUs
Premium RS pool 250 DTU	Business Critical 2 vCores (Gen4)	Premium pool 250 eDTUs
Premium RS pool 500 DTU	Business Critical 4 vCores (Gen4)	Premium pool 500 eDTUs
Premium RS pool 1000 DTU	Business Critical 8 vCores (Gen4)	Premium pool 1000 eDTUs

Take advantage of our new offers

Our service tiers in the vCore-based purchasing model are eligible for special offers that can save you up to 80% versus license-included pricing. Use your SQL Server Standard or Enterprise edition licenses with active Software Assurance to save up to 55% versus license-included pricing with the [Azure Hybrid Benefit for SQL Server](#). You can combine the hybrid benefit with [Azure SQL Database reserved capacity](#) pricing and save up to 80% when you commit upfront to a one or three-year term. Activate both benefits today from Azure portal.

If you have any questions or concerns regarding this change or if you require migration assistance, contact [Microsoft](#).

Migration from a Premium RS service tier to a service tier in either the DTU or the vCore model

Migration of a database

Migrating a database from a Premium RS service tier to a service tier in either the DTU or the vCore model is similar to upgrading or downgrading between service tiers in the Premium RS service tier.

Using database copy to convert a Premium RS database to a DTU-based or vCore-based database

You can copy any database with a Premium RS compute size to a database with a DTU-based or vCore-based compute size without restrictions or special sequencing as long as the target compute size supports the maximum database size of the source database. The database copy creates a snapshot of data as of the starting time of the copy operation and does not perform data synchronization between the source and the target.

Next steps

- For details on specific compute sizes and storage size choices available for single databases, see [SQL Database vCore-based resource limits for single databases](#)
- For details on specific compute sizes and storage size choices available for elastic pools, see [SQL Database vCore-based resource limits for elastic pools](#).

Getting started with single databases in Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

A single database is fully managed PaaS database as a service (DbaaS) that is ideal storage engine for the modern cloud-born applications. In this section, you will learn how to quickly configure and create a single database in SQL Database.

Quickstart overview

In this section, you will see an overview of available articles that can help you to quickly get started with single databases. The following quickstarts enable you to quickly create a single database, configure a database server firewall rule, and then import a database into the new single database using a `.bacpac` file:

- [Create a single database using the Azure portal](#).
- After creating the database, you would need to [secure your database by configuring firewall rules](#).
- If you have an existing database on SQL Server that you want to migrate to Azure, you should install [Data Migration Assistant \(DMA\)](#) that will analyze your databases on SQL Server and find any issue that could block migration to the single database deployment option. If you don't find any issue, you can export your database as `.bacpac` file and [import it using the Azure portal or SqlPackage](#).

Automating management operations

You can use PowerShell or the Azure CLI to create, configure, and scale your database.

- [Create and configure a single database using PowerShell](#)
- [Create and configure a single database using Azure CLI](#)
- [Update your single database and scale resources using PowerShell](#)
- [Update your single database and scale resources using Azure CLI](#)

Migrating to a single database with minimal downtime

These quickstarts enable you to quickly create or import your database to Azure using a `.bacpac` file. However, `.bacpac` and `.dacpac` files are designed to quickly move databases across different versions of SQL Server and deployment options within Azure SQL Database, or to implement continuous integration in your DevOps pipeline. However, this method is not designed for migration of your production databases with minimal downtime, because you would need to stop adding new data, wait for the export of the source database to a `.bacpac` file to complete, and then wait for the import into Azure SQL Database to complete. All of this waiting results in downtime of your application, especially for large databases. To move your production database, you need a better way to migrate that guarantees minimal downtime of migration. For this, use the [Data Migration Service \(DMS\)](#) to migrate your database with the minimal downtime.. DMS accomplishes this by incrementally pushing the changes made in your source database to the single database being restored. This way, you can quickly switch your application from source to target database with the minimal downtime.

Hands-on learning modules

The following Microsoft Learn modules help you learn for free about Azure SQL Database.

- [Provision an Azure SQL database to store application data](#)

- Develop and configure an ASP.NET application that queries an Azure SQL Database
- Secure your Azure SQL Database

Next steps

- Find a [high-level list of supported features in Azure SQL Database](#).
- Learn how to make your [database more secure](#).
- Find more advanced how-to's in [how to use a single database in Azure SQL Database](#).
- Find more sample scripts written in [PowerShell](#) and [Azure CLI](#).
- Learn more about the [management API](#) that you can use to configure your databases.
- [Identify the right Azure SQL Database/Managed Instance SKU for your on-premises database](#).

Getting started with Azure SQL Database managed instance

11/7/2019 • 6 minutes to read • [Edit Online](#)

The [managed instance](#) deployment option creates a database with near 100% compatibility with the latest SQL Server on-premises (Enterprise Edition) database engine, providing a native [virtual network \(VNet\)](#) implementation that addresses common security concerns, and a [business model](#) favorable for on-premises SQL Server customers. In this article, you will learn how to quickly configure and create a managed instance and migrate your databases.

Quickstart overview

The following quickstarts enable you to quickly create a managed instance, configure a virtual machine or point to site VPN connection for client application, and restore a database to your new managed instance using a `.bak` file.

Configure environment

As a first step, you would need to create your first Managed Instance with the network environment where it will be placed, and enable connection from the computer or virtual machine where you are executing queries to Managed Instance. You can use the following guides:

- [Create a managed instance using the Azure portal](#). In the Azure portal, you configure the necessary parameters (username/password, number of cores, and max storage amount), and automatically create the Azure network environment without the need to know about networking details and infrastructure requirements. You just make sure that you have a [subscription type](#) that is currently allowed to create a managed instance. If you have your own network that you want to use or you want to customize the network, see [configure an existing virtual network for Azure SQL Database managed instance](#) or [create a virtual network for Azure SQL Database managed instance](#).
- A managed instance is created in own VNet with no public endpoint. For client application access, you can either **create a VM in the same VNet (different subnet)** or **create a point-to-site VPN connection to the VNet from your client computer** using one of these quickstarts:
 - Enable [public endpoint](#) on your Managed Instance in order to access your data directly from your environment.
 - Create [Azure Virtual Machine in the managed instance VNet](#) for client application connectivity, including SQL Server Management Studio.
 - Set up [point-to-site VPN connection to your managed instance](#) from your client computer on which you have SQL Server Management Studio and other client connectivity applications. This is other of two options for connectivity to your managed instance and to its VNet.

NOTE

You can also use express route or site-to-site connection from your local network, but these approaches are out of the scope of these quickstarts.

As an alternative to manual creation of Managed Instance, you can use [PowerShell](#), [PowerShell with Resource Manager template](#), or [Azure CLI](#) to script and automate this process.

Migrate your databases

After you create a managed instance and configure access, you can start migrating your databases from SQL Server on-premises or Azure VMs. Migration fails if you have some unsupported features in the source database that you want to migrate. To avoid failures and check compatibility, you can install [Data Migration Assistant \(DMA\)](#) that analyzes your databases on SQL Server and finds any issue that could block migration to a managed instance, such as existence of [FileStream](#) or multiple log files. If you resolve these issues, your databases are ready to migrate to managed instance. [Database Experimentation Assistant](#) is another useful tool that can record your workload on SQL Server and replay it on a managed instance so you can determine are there going to be any performance issues if you migrate to a managed instance.

Once you are sure that you can migrate your database to a managed instance, you can use the native SQL Server restore capabilities to restore a database into a managed instance from a `.bak` file. You can use this method to migrate databases from SQL Server database engine installed on-premises or Azure VM. For a quickstart, see [Restore from backup to a managed instance](#). In this quickstart, you restore from a `.bak` file stored in Azure Blob storage using the `RESTORE` Transact-SQL command.

TIP

To use the `BACKUP` Transact-SQL command to create a backup of your database in Azure Blob storage, see [SQL Server backup to URL](#).

These quickstarts enable you to quickly create, configure, and restore database backup to a managed instance. In some scenarios, you would need to customize or automate deployment of managed instances and the required networking environment. These scenarios will be described below.

Customize network environment

Although the VNet/subnet can be automatically configured when the instance is created using [the Azure portal](#), it might be good to create it before you start creating Managed Instances because you can configure the parameters of VNet and subnet. The easiest way to create and configure the network environment is to use [Azure Resource deployment](#) template that will create and configure you network and subnet where the instance will be placed. You just need to press the Azure Resource Manager deploy button and populate the form with parameters.

As an alternative, you can also use this [PowerShell script](#) to automate creation of the network.

If you already have a VNet and subnet where you would like to deploy your managed instance, you need to make sure that your VNet and subnet satisfy the [networking requirements](#). Use this [PowerShell script to verify that your subnet is properly configured](#). This script validates your network and report any issues, and it tells you what should be changed and then offers to make the necessary changes in your VNet/subnet. Run this script if you don't want to configure your VNet/subnet manually. You can also run it after any major reconfiguration of your network infrastructure. If you want to create and configure your own network, read [connectivity architecture](#) and this [ultimate guide for creating and configuring a managed instance environment](#).

Migrate to a managed instance

Articles in these quickstarts enable you to quickly set up a managed instance and move your databases using the native `RESTORE` capability. This is a good starting point if you want to complete quick proof-of concepts and verify that your solution can work on Managed Instance.

However, in order to migrate your production database or even dev/test databases that you want to use for some performance test, you would need to consider using some additional techniques, such as:

- Performance testing - You should measure baseline performance on your source SQL Server instance and compare them with the performance on the destination Managed Instance where you have migrated the database. Learn more about the [best practices for performance comparison](#).

- Online migration - With the native `RESTORE` described in this article, you have to wait for the databases to be restored (and copied to Azure Blob storage if not already stored there). This causes some downtime of your application especially for larger databases. To move your production database, use the [Data Migration service \(DMS\)](#) to migrate your database with the minimal downtime. DMS accomplishes this by incrementally pushing the changes made in your source database to the managed instance database being restored. This way, you can quickly switch your application from source to target database with the minimal downtime.

Learn more about the [recommended migration process](#).

Next steps

- Find a [high-level list of supported features in managed instance here](#) and [details and known issues here](#).
- Learn about [technical characteristics of managed instance](#).
- Find more advanced how-to's in [how to use a managed instance in Azure SQL Database](#).
- [Identify the right Azure SQL Database/Managed Instance SKU for your on-premises database](#).

Azure SQL Database Features

1/2/2020 • 13 minutes to read • [Edit Online](#)

Azure SQL Database service shares a common code base with the latest stable version of SQL Server. Most of the standard SQL language, query processing, and database management features are identical in SQL Server and Azure SQL Database. The features that are common in SQL Server and all flavors of Azure SQL Database are:

- Language features - [Control of flow language keywords](#), [Cursors](#), [Data types](#), [DML statements](#), [Predicates](#), [Sequence numbers](#), [Stored procedures](#), and [Variables](#).
- Database features - [Automatic tuning \(plan forcing\)](#), [Change tracking](#), [Database collation](#), [Contained databases](#), [Contained users](#), [Data compression](#), [Database configuration settings](#), [Online index operations](#), [Partitioning](#), and [Temporal tables](#) (see [getting started guide](#)).
- Security features - [Application roles](#), [Dynamic data masking](#) (see [getting started guide](#)), [Row Level Security](#), and [Threat detection](#) - see [getting started guides](#) for [single database](#) and [elastic pools](#) and [managed instance](#).
- Multi-model capabilities - [Graph processing](#), [JSON data](#) (see [getting started guide](#)), [OPENXML](#), [Spatial](#), [OPENJSON](#), and [XML indexes](#).

Azure SQL Database manages your databases and guarantees their high-availability. Some features that might affect high-availability or cannot be used in PaaS world have limited functionalities in Azure SQL Database. In addition, some database features depend on the type of Azure SQL Database that you create. These features are described in the tables below. With Azure SQL Database, you can create a database as part of a [managed instance](#), as a single database, or as part of an elastic pool. If you need more details about the differences, you can find them in the separate pages for [Single database and Elastic pools](#) or [Managed Instance](#).

SQL features

The following table lists the major features of SQL Server and provides information about whether the feature is partially or fully supported in Managed Instance or Single Database and Elastic pools, with a link to more information about the feature.

SQL FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Always Encrypted	Yes - see Cert store and Key vault	Yes - see Cert store and Key vault
Always On Availability Groups	99.99-99.995% availability is guaranteed for every database. Disaster recovery is discussed in Overview of business continuity with Azure SQL Database	99.99% availability is guaranteed for every database and cannot be managed by user . Disaster recovery is discussed in Overview of business continuity with Azure SQL Database . Use Auto-failover groups to configure secondary Always On Managed Instance in another region. Other SQL Server instances and Single databases cannot be used as secondaries for Managed Instance.
Attach a database	No	No

SQL FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Auditing	Yes	Yes, with some differences
Azure Active Directory (AAD) authentication	Yes. AAD users only.	Yes. Including server-level AAD logins.
BACKUP command	No, only system-initiated automatic backups - see Automated backups	Yes, user initiated copy-only backups to Azure Blob Storage (automatic system backups cannot be initiated by user) - see Backup differences
Built-in functions	Most - see individual functions	Yes - see Stored procedures, functions, triggers differences
BULK INSERT statement	Yes, but just from Azure Blob storage as a source.	Yes, but just from Azure Blob Storage as a source - see differences .
Certificates and asymmetric keys	Yes, without access to file system for <code>BACKUP</code> and <code>CREATE</code> operations.	Yes, without access to file system for <code>BACKUP</code> and <code>CREATE</code> operations - see certificate differences .
Change data capture - CDC	No	Yes
Collation - server/instance	No, default logical server collation <code>SQL_Latin1_General_CI_AS</code> is always used.	Yes, can be set when the instance is created and cannot be updated later.
Columnstore indexes	Yes - Premium tier, Standard tier - S3 and above, General Purpose tier, Business Critical, and HyperScale tiers	Yes
Common language runtime - CLR	No	Yes, but without access to file system in <code>CREATE ASSEMBLY</code> statement - see CLR differences
Credentials	Yes, but only database scoped credentials .	Yes, but only Azure Key Vault and <code>SHARED ACCESS SIGNATURE</code> are supported see details
Cross-database/three-part name queries	No - see Elastic queries	Yes, plus Elastic queries
Cross-database transactions	No	Yes, within the instance. See Linked server differences for cross-instance queries.
Database mail - DbMail	No	Yes
Database mirroring	No	No
Database snapshots	No	No
DBCC statements	Most - see individual statements	Yes - see DBCC differences

SQL FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
DDL statements	Most - see individual statements	Yes - see T-SQL differences
DDL triggers	Database only	Yes
Distributed partition views	No	Yes
Distributed transactions - MS DTC	No - see Elastic transactions	No - see Linked server differences . Try to consolidate databases from several distributed SQL Server instances into one managed instance during migration.
DML triggers	Most - see individual statements	Yes
DMVs	Most - see individual DMVs	Yes - see T-SQL differences
Event notifications	No - see Alerts	No
Expressions	Yes	Yes
Extended events (XEvent)	Some - see Extended events in SQL Database	Yes - see Extended events differences
Extended stored procedures	No	No
Files and file groups	Primary file group only	Yes. File paths are automatically assigned and the file location cannot be specified in <code>ALTER DATABASE ADD FILE</code> statement.
Filestream	No	No
Full-text search (FTS)	Yes, but third-party word breakers are not supported	Yes, but third-party word breakers are not supported
Functions	Most - see individual functions	Yes - see Stored procedures, functions, triggers differences
In-memory optimization	Yes - Premium and Business Critical tiers only Limited support for non-persistent In-Memory objects such as table types	Yes - Business Critical tier only
Language elements	Most - see individual elements	Yes - see T-SQL differences
Linked servers	No - see Elastic query	Yes. Only to SQL Server and SQL Database without distributed transactions.

SQL FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Linked servers that read from files (CSV, Excel)	No. Use BULK INSERT or OPENROWSET as an alternative for CSV format.	No. Use BULK INSERT or OPENROWSET as an alternative for CSV format. Track this requests on Managed Instance Feedback item
Log shipping	High availability is included with every database. Disaster recovery is discussed in Overview of business continuity with Azure SQL Database	Natively built in as a part of DMS migration process. Not available as High availability solution, because other High availability methods are included with every database and it is not recommended to use Log-shipping as HA alternative. Disaster recovery is discussed in Overview of business continuity with Azure SQL Database . Not available as a replication mechanism between databases - use secondary replicas on Business Critical tier , auto-failover groups , or transactional replication as the alternatives.
Logins and users	Yes, but <code>CREATE</code> and <code>ALTER</code> login statements do not offer all the options (no Windows and server-level Azure Active Directory logins). <code>EXECUTE AS LOGIN</code> is not supported - use <code>EXECUTE AS USER</code> instead.	Yes, with some differences . Windows logins are not supported and they should be replaced with Azure Active Directory logins.
Minimal logging in bulk import	No, only Full Recovery model is supported.	No, only Full Recovery model is supported.
Modifying system data	No	Yes
OLE Automation	No	No
OPENDATASOURCE	No	Yes, only to other Azure SQL Databases and SQL Servers. See T-SQL differences
OPENQUERY	No	Yes, only to other Azure SQL Databases and SQL Servers. See T-SQL differences
OPENROWSET	Yes, only to import from Azure Blob storage.	Yes, only to other Azure SQL Databases and SQL Servers, and to import from Azure Blob storage. See T-SQL differences
Operators	Most - see individual operators	Yes - see T-SQL differences
Polybase	No. You can query data in the files placed on Azure Blob Storage using <code>OPENROWSET</code> function.	No. You can query data in the files placed on Azure Blob Storage using <code>OPENROWSET</code> function.
Query Notifications	No	Yes

SQL FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Machine Learning Services (Formerly R Services)	Yes, in public preview	No
Recovery models	Only Full Recovery that guarantees high availability is supported. Simple and Bulk Logged recovery models are not available.	Only Full Recovery that guarantees high availability is supported. Simple and Bulk Logged recovery models are not available.
Resource governor	No	Yes
RESTORE statements	No	Yes, with mandatory FROM URL options for the backups files placed on Azure Blob Storage. See Restore differences
Restore database from backup	From automated backups only - see SQL Database recovery	From automated backups - see SQL Database recovery and from full backups placed on Azure Blob Storage - see Backup differences
Restore database to SQL Server	No. Use BACPAC or BCP instead of native restore.	No, because SQL Server Database Engine used in Managed Instance has higher version than any RTM version of SQL Server used on-premises. Use BACPAC, BCP, or Transactional replication instead.
Semantic search	No	No
Service Broker	No	Yes, but only within the instance. If you are using remote Service Broker routes, try to consolidate databases from several distributed SQL Server instances into one managed instance during migration and use only local routes. See Service Broker differences
Server configuration settings	No	Yes - see T-SQL differences
Set statements	Most - see individual statements	Yes - see T-SQL differences
SQL Server Agent	No - see Elastic jobs	Yes - see SQL Server Agent differences
SQL Server Auditing	No - see SQL Database auditing	Yes - see Auditing differences
System stored functions	Most - see individual functions	Yes - see Stored procedures, functions, triggers differences
System stored procedures	Some - see individual stored procedures	Yes - see Stored procedures, functions, triggers differences
System tables	Some - see individual tables	Yes - see T-SQL differences
System catalog views	Some - see individual views	Yes - see T-SQL differences

SQL FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
TempDB	Yes. 32GB size per core for every database.	Yes. 24GB size per vCore for entire GP tier and limited by instance size on BC tier
Temporary tables	Local and database-scoped global temporary tables	Local and instance-scoped global temporary tables
Time zone choice	No	Yes, and it must be configured when the Managed Instance is created.
Trace flags	No	Yes, but only limited set of global trace flags. See DBCC differences
Transactional Replication	Yes, Transactional and snapshot replication subscriber only	Yes, in public preview . See the constraints here .
Transparent data encryption (TDE)	Yes - General Purpose and Business Critical service tiers only	Yes
Windows authentication	No	No
Windows Server Failover Clustering	No. Other techniques that provide high availability are included with every database. Disaster recovery is discussed in Overview of business continuity with Azure SQL Database	No. Other techniques that provide high availability are included with every database. Disaster recovery is discussed in Overview of business continuity with Azure SQL Database

Platform capabilities

Azure platform provides a number of PaaS capabilities that are added as an additional value to the standard Database features. There is a number of external services that can be used with Azure SQL Database service.

PLATFORM FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Active geo-replication	Yes - all service tiers other than hyperscale	No, see Auto-failover groups as an alternative
Auto-failover groups	Yes - all service tiers other than hyperscale	Yes, see Auto-failover groups
Auto-scale	Yes, but only in serverless model . In the non-serverless model, the change of service tier (change of vCore, storage, or DTU) is fast and online. The service tier change requires minimal or no downtime.	No, you need to choose reserved compute and storage. The change of service tier (vCore or max storage) is online and requires minimal or no downtime.
Automatic backups	Yes. Full backups are taken every 7 days, differential 12 hours, and log backups every 5-10 min.	Yes. Full backups are taken every 7 days, differential 12 hours, and log backups every 5-10 min.
Automatic tuning (indexes)	Yes	No

PLATFORM FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Availability Zones	Yes	No
Azure Resource Health	Yes	No
Backup retention	Yes. 7 days default, max 35 days.	Yes. 7 days default, max 35 days.
Data Migration Service (DMS)	Yes	Yes
File system access	No. Use BULK INSERT or OPENROWSET to access and load data from Azure Blob Storage as an alternative.	No. Use BULK INSERT or OPENROWSET to access and load data from Azure Blob Storage as an alternative.
Geo-restore	Yes - all service tiers other than hyperscale	Yes - all service tiers other than hyperscale
Hyperscale architecture	Yes	No
Long-term backup retention - LTR	Yes, keep automatically taken backups up to 10 years.	Not yet. Use COPY_ONLY manual backups as a temporary workaround.
Pause/resume	Yes, in serverless model	No
Policy-based management	No	No
Public IP address	Yes. The access can be restricted using firewall or service endpoints.	Yes. Needs to be explicitly enabled and port 3342 must be enabled in NSG rules. Public IP can be disabled if needed. See Public endpoint for more details.
Point in time database restore	Yes - all service tiers other than hyperscale - see SQL Database recovery	Yes - see SQL Database recovery
Resource pools	Yes, as Elastic pools	Yes. A single managed instance can have multiple databases that share the same pool of resources. In addition, you can deploy multiple managed instances in instance pools(Preview) that can share the resources.
Scaling up or down (online)	Yes, you can either change DTU or reserved vCores or max storage with the minimal downtime.	Yes, you can change reserved vCores or max storage with the minimal downtime.
SQL Alias	No, use DNS Alias	No, use Cliconfg to set up alias on the client machines.
SQL Analytics	Yes	Yes
SQL Data Sync	Yes	No

PLATFORM FEATURE	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
SQL Server Analysis Services (SSAS)	No, Azure Analysis Services is a separate Azure cloud service.	No, Azure Analysis Services is a separate Azure cloud service.
SQL Server Integration Services (SSIS)	<p>Yes, with a managed SSIS in Azure Data Factory (ADF) environment, where packages are stored in SSISDB hosted by Azure SQL Database and executed on Azure SSIS Integration Runtime (IR), see Create Azure-SSIS IR in ADF.</p> <p>To compare the SSIS features in SQL Database server and Managed Instance, see Compare an Azure SQL Database single database, elastic pool, and managed instance.</p>	<p>Yes, with a managed SSIS in Azure Data Factory (ADF) environment, where packages are stored in SSISDB hosted by Managed Instance and executed on Azure SSIS Integration Runtime (IR), see Create Azure-SSIS IR in ADF.</p> <p>To compare the SSIS features in SQL Database and Managed Instance, see Compare an Azure SQL Database single database, elastic pool, and managed instance.</p>
SQL Server Reporting Services (SSRS)	No - see Power BI	No - see Power BI
Query Performance Insights (QPI)	Yes	No. Use built-in reports in SQL Server Management Studio and Azure Data Studio.
VNet	Partial, it enables restricted access using VNet Endpoints	Yes, Managed Instance is injected in customer's VNet. See subnet and VNet
VNet Service endpoint	Yes	No
VNet Global peering	Yes, using Private IP and service endpoints	No, Managed Instance is not supported due to Load balancer constraint in VNet global peering.

Tools

Azure SQL database supports various data tools that can help you to manage your data.

TOOL	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Azure portal	Yes	Yes
Azure CLI	Yes	Yes
Azure Data Studio	Yes	Yes
Azure Powershell	Yes	Yes
BACPAC file (export)	Yes - see SQL Database export	Yes - see SQL Database export
BACPAC file (import)	Yes - see SQL Database import	Yes - see SQL Database import
Data Quality Services (DQS)	No	No

TOOL	SINGLE DATABASES AND ELASTIC POOLS	MANAGED INSTANCES AND INSTANCE POOLS
Master Data Services (MDS)	No	No
SMO	Yes	Yes version 150
SQL Server Data Tools (SSDT)	Yes	Yes
SQL Server Management Studio (SSMS)	Yes	Yes version 18.0 and higher
SQL Server PowerShell	Yes	Yes
SQL Server Profiler	No - see Extended events	Yes
System Center Operations Manager (SCOM)	Yes	Yes, in preview

Migration methods

You can use different migration methods to move your data between SQL Server, Single Database, and Managed Instance databases. Some methods are **Online** and picking-up all changes that are made on the source while you are running migration, while in **Offline** methods you need to stop your workload that is modifying data on the source while the migration is in progress.

SOURCE	SINGLE DATABASE AND ELASTIC POOL	MANAGED INSTANCE AND INSTANCE POOLS
SQL Server (on-prem, AzureVM, Amazon RDS)	Online: Data Migration Service (DMS) , Transactional Replication Offline: BACPAC file (import) , BCP	Online: Data Migration Service (DMS) , Transactional Replication Offline: Native backup/restore, BACPAC file (import) , BCP, Snapshot replication
Single database	Offline: BACPAC file (import) , BCP	Offline: BACPAC file (import) , BCP
Managed Instance	Online: Transactional Replication Offline: BACPAC file (import) , BCP, Snapshot replication	Online: Transactional Replication Offline: Cross-instance point-in-time restore (Azure PowerShell or Azure CLI), Native backup/restore, BACPAC file (import) , BCP, Snapshot replication

Next steps

Microsoft continues to add features to Azure SQL Database. Visit the Service Updates webpage for Azure for the newest updates using these filters:

- Filtered to the [SQL Database service](#).
- Filtered to General Availability (GA) announcements for SQL Database features.

For more information about the Azure SQL Database flavors, see:

- [What is SQL Database?](#)
- [What is a Managed Instance?](#)

- What are Managed Instance pools?

How to use Azure SQL Database

11/14/2019 • 2 minutes to read • [Edit Online](#)

In this section you can find various guides, scripts, and explanations that can help you to manage and configure your Azure SQL Database. You can also find specific how-to guides for [single database](#) and [Managed Instance](#).

Load data

- [Copy a single database or pooled database within Azure](#)
- [Import a DB from a BACPAC](#)
- [Export a DB to BACPAC](#)
- [Load data with BCP](#)
- [Load data with ADF](#)

Data sync

- [SQL Data Sync](#)
- [Data Sync Agent](#)
- [Replicate schema changes](#)
- [Monitor with OMS](#)
- [Best practices for Data Sync](#)
- [Troubleshoot Data Sync](#)

Monitoring and tuning

- [Manual tuning](#)
- [Use DMVs to monitor performance](#)
- [Use Query store to monitor performance](#)
- [Troubleshoot performance with Intelligent Insights](#)
- [Use Intelligent Insights diagnostics log](#)
- [Monitor In-memory OLTP space](#)

Extended events

- [Extended events](#)
- [Store Extended events into event file](#)
- [Store Extended events into ring buffer](#)

Configure features

- [Configure Azure AD auth](#)
- [Configure Conditional Access](#)
- [Multi-factor AAD auth](#)
- [Configure multi-factor auth](#)
- [Configure temporal retention policy](#)
- [Configure TDE with BYOK](#)
- [Rotate TDE BYOK keys](#)
- [Remove TDE protector](#)
- [Configure In-Memory OLTP](#)

- [Configure Azure Automation](#)

Develop applications

- [Connectivity](#)
- [Use Spark Connector](#)
- [Authenticate app](#)
- [Use batching for better performance](#)
- [Connectivity guidance](#)
- [DNS aliases](#)
- [Setup DNS alias PowerShell](#)
- [Ports - ADO.NET](#)
- [C and C ++](#)
- [Excel](#)

Design applications

- [Design for disaster recovery](#)
- [Design for elastic pools](#)
- [Design for app upgrades](#)

Design Multi-tenant SaaS applications

- [SaaS design patterns](#)
- [SaaS video indexer](#)
- [SaaS app security](#)

Next steps

- Learn more about [How-to guides for managed instances](#).
- Learn more about [How-to guides for single databases](#).

An overview of Azure SQL Database security capabilities

1/14/2020 • 9 minutes to read • [Edit Online](#)

This article outlines the basics of securing the data tier of an application using Azure SQL Database. The security strategy described follows the layered defense-in-depth approach as shown in the picture below, and moves from the outside in:



Network security

Microsoft Azure SQL Database provides a relational database service for cloud and enterprise applications. To help protect customer data, firewalls prevent network access to the database server until access is explicitly granted based on IP address or Azure Virtual network traffic origin.

IP firewall rules

IP firewall rules grant access to databases based on the originating IP address of each request. For more information, see [Overview of Azure SQL Database and SQL Data Warehouse firewall rules](#).

Virtual network firewall rules

[Virtual network service endpoints](#) extend your virtual network connectivity over the Azure backbone and enable Azure SQL Database to identify the virtual network subnet that traffic originates from. To allow traffic to reach Azure SQL Database, use the SQL [service tags](#) to allow outbound traffic through Network Security Groups.

[Virtual network rules](#) enable Azure SQL Database to only accept communications that are sent from selected subnets inside a virtual network.

NOTE

Controlling access with firewall rules does *not* apply to a **managed instance**. For more information about the networking configuration needed, see [connecting to a managed instance](#)

Access management

IMPORTANT

Managing databases and database servers within Azure is controlled by your portal user account's role assignments. For more information on this article, see [Role-based access control in Azure portal](#).

Authentication

Authentication is the process of proving the user is who they claim to be. Azure SQL Database supports two types of authentication:

- **SQL authentication:**

SQL database authentication refers to the authentication of a user when connecting to [Azure SQL Database](#) using username and password. During the database server creation for the database, a "Server admin" login with a username and password must be specified. Using these credentials, a "server admin" can authenticate to any database on that database server as the database owner. After that, additional SQL logins and users can be created by the server admin, which enable users to connect using username and password.

- **Azure Active Directory authentication:**

Azure Active Directory authentication is a mechanism of connecting to [Azure SQL Database](#) and [SQL Data Warehouse](#) by using identities in Azure Active Directory (Azure AD). Azure AD authentication allows administrators to centrally manage the identities and permissions of database users along with other Microsoft services in one central location. This includes the minimization of password storage and enables centralized password rotation policies.

A server admin called the **Active Directory administrator** must be created to use Azure AD authentication with SQL Database. For more information, see [Connecting to SQL Database By Using Azure Active Directory Authentication](#). Azure AD authentication supports both managed and federated accounts. The federated accounts support Windows users and groups for a customer domain federated with Azure AD.

Additional Azure AD authentication options available are [Active Directory Universal Authentication for SQL Server Management Studio connections](#) including [Multi-Factor Authentication](#) and [Conditional Access](#).

IMPORTANT

Managing databases and servers within Azure is controlled by your portal user account's role assignments. For more information on this article, see [Role-based access control in Azure portal](#). Controlling access with firewall rules does *not* apply to **a managed instance**. Please see the following article on [connecting to a managed instance](#) for more information about the networking configuration needed.

Authorization

Authorization refers to the permissions assigned to a user within an Azure SQL Database, and determines what the user is allowed to do. Permissions are controlled by adding user accounts to [database roles](#) and assigning database-level permissions to those roles or by granting the user certain [object-level permissions](#). For more information, see [Logins and users](#)

As a best practice, create custom roles when needed. Add users to the role with the least privileges required to do their job function. Do not assign permissions directly to users. The server admin account is a member of the built-

in db_owner role, which has extensive permissions and should only be granted to few users with administrative duties. For Azure SQL Database applications, use the [EXECUTE AS](#) to specify the execution context of the called module or use [Application Roles](#) with limited permissions. This practice ensures that the application that connects to the database has the least privileges needed by the application. Following these best practices also fosters separation of duties.

Row-level security

Row-Level Security enables customers to control access to rows in a database table based on the characteristics of the user executing a query (for example, group membership or execution context). Row-Level Security can also be used to implement custom Label-based security concepts. For more information, see [Row-Level security](#).



Threat protection

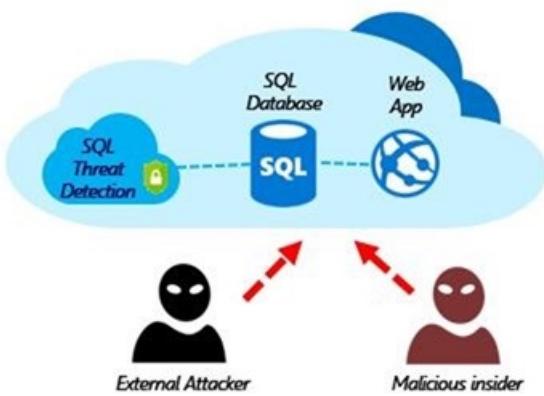
SQL Database secures customer data by providing auditing and threat detection capabilities.

SQL auditing in Azure Monitor logs and Event Hubs

SQL Database auditing tracks database activities and helps to maintain compliance with security standards by recording database events to an audit log in a customer-owned Azure storage account. Auditing allows users to monitor ongoing database activities, as well as analyze and investigate historical activity to identify potential threats or suspected abuse and security violations. For more information, see [Get started with SQL Database Auditing](#).

Advanced Threat Protection

Advanced Threat Protection is analyzing your SQL Server logs to detect unusual behavior and potentially harmful attempts to access or exploit databases. Alerts are created for suspicious activities such as SQL injection, potential data infiltration, and brute force attacks or for anomalies in access patterns to catch privilege escalations and breached credentials use. Alerts are viewed from the [Azure Security Center](#), where the details of the suspicious activities are provided and recommendations for further investigation given along with actions to mitigate the threat. Advanced Threat Protection can be enabled per server for an additional fee. For more information, see [Get started with SQL Database Advanced Threat Protection](#).



Information protection and encryption

Transport Layer Security TLS (Encryption-in-transit)

SQL Database secures customer data by encrypting data in motion with [Transport Layer Security](#).

Sql Server enforces encryption (SSL/TLS) at all times for all connections. This ensures all data is encrypted "in transit" between the client and server irrespective of the setting of **Encrypt** or **TrustServerCertificate** in the connection string.

As a best practice, recommend that in your application's connection string you specify an encrypted connection and **not** trust the server certificate. This forces your application to verify the server certificate and thus prevents your application from being vulnerable to man in the middle type attacks.

For example when using the ADO.NET driver this is accomplished via **Encrypt=True** and **TrustServerCertificate=False**. If you obtain your connection string from the Azure portal, it will have the correct settings.

IMPORTANT

Note that some non-Microsoft drivers may not use TLS by default or rely on an older version of TLS (<1.2) in order to function. In this case SQL Server still allows you to connect to your database. However, we recommend that you evaluate the security risks of allowing such drivers and application to connect to SQL Database, especially if you store sensitive data.

For further information about TLS and connectivity, see [TLS considerations](#)

Transparent Data Encryption (Encryption-at-rest)

[Transparent Data Encryption \(TDE\) for Azure SQL Database](#) adds a layer of security to help protect data at rest from unauthorized or offline access to raw files or backups. Common scenarios include datacenter theft or unsecured disposal of hardware or media such as disk drives and backup tapes. TDE encrypts the entire database using an AES encryption algorithm, which doesn't require application developers to make any changes to existing applications.

In Azure, all newly created SQL databases are encrypted by default and the database encryption key is protected by a built-in server certificate. Certificate maintenance and rotation are managed by the service and requires no input from the user. Customers who prefer to take control of the encryption keys can manage the keys in [Azure Key Vault](#).

Key management with Azure Key Vault

[Bring Your Own Key](#) (BYOK) support for [Transparent Data Encryption](#) (TDE) allows customers to take ownership of key management and rotation using [Azure Key Vault](#), Azure's cloud-based external key management system. If the database's access to the key vault is revoked, a database cannot be decrypted and read into memory. Azure Key Vault provides a central key management platform, leverages tightly monitored hardware security modules (HSMs), and enables separation of duties between management of keys and data to help meet security

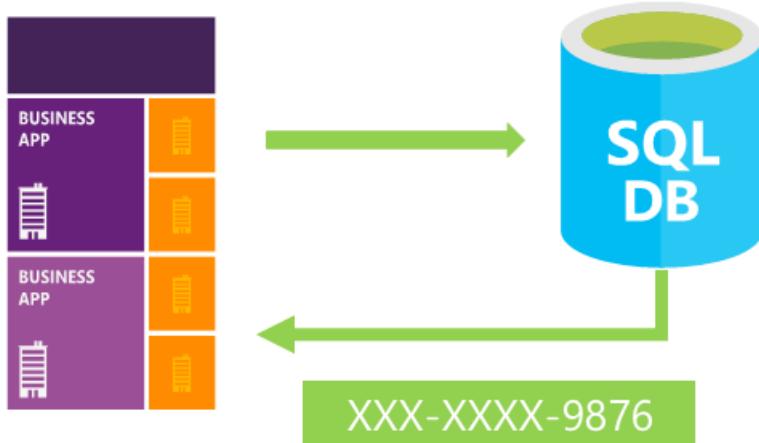
compliance requirements.

Always Encrypted (Encryption-in-use)



[Always Encrypted](#) is a feature designed to protect sensitive data stored in specific database columns from access (for example, credit card numbers, national identification numbers, or data on a *need to know* basis). This includes database administrators or other privileged users who are authorized to access the database to perform management tasks, but have no business need to access the particular data in the encrypted columns. The data is always encrypted, which means the encrypted data is decrypted only for processing by client applications with access to the encryption key. The encryption key is never exposed to SQL and can be stored either in the [Windows Certificate Store](#) or in [Azure Key Vault](#).

Dynamic data masking



SQL Database dynamic data masking limits sensitive data exposure by masking it to non-privileged users. Dynamic data masking automatically discovers potentially sensitive data in Azure SQL Database and provides actionable recommendations to mask these fields, with minimal impact on the application layer. It works by obfuscating the sensitive data in the result set of a query over designated database fields, while the data in the database is not changed. For more information, see [Get started with SQL Database dynamic data masking](#).

Security management

Vulnerability assessment

[Vulnerability assessment](#) is an easy to configure service that can discover, track, and help remediate potential database vulnerabilities with the goal to proactively improve overall database security. Vulnerability assessment (VA) is part of the advanced data security (ADS) offering, which is a unified package for advanced SQL security capabilities. Vulnerability assessment can be accessed and managed via the central SQL ADS portal.

Data discovery & classification

Data discovery & classification (currently in preview) provides advanced capabilities built into Azure SQL Database for discovering, classifying, labeling, and protecting the sensitive data in your databases. Discovering and classifying your utmost sensitive data (business/financial, healthcare, personal data, etc.) can play a pivotal role in your organizational Information protection stature. It can serve as infrastructure for:

- Various security scenarios, such as monitoring (auditing) and alerting on anomalous access to sensitive data.
- Controlling access to, and hardening the security of, databases containing highly sensitive data.
- Helping meet data privacy standards and regulatory compliance requirements.

For more information, see [Get started with data discovery & classification](#).

Compliance

In addition to the above features and functionality that can help your application meet various security requirements, Azure SQL Database also participates in regular audits, and has been certified against a number of compliance standards. For more information, see the [Microsoft Azure Trust Center](#) where you can find the most current list of SQL Database compliance certifications.

Next steps

- For a discussion of the use of access control features in SQL Database, see [Control access](#).
- For a discussion of database auditing, see [SQL Database auditing](#).
- For a discussion of threat detection, see [SQL Database threat detection](#).

Playbook for Addressing Common Security Requirements with Azure SQL Database

2/20/2020 • 38 minutes to read • [Edit Online](#)

NOTE

This document provides best practices on how to solve common security requirements. Not all requirements are applicable to all environments, and you should consult your database and security team on which features to implement.

Solving common security requirements

This document provides guidance on how to solve common security requirements for new or existing applications using Azure SQL Database. It's organized by high level security areas. For addressing specific threats, refer to the [Common security threats and potential mitigations](#) section. Although some of the presented recommendations are applicable when migrating applications from on-premises to Azure, migration scenarios are not the focus of this document.

Azure SQL Database deployment offers covered in this guide

- [SQL Databases](#): single databases and [elastic pools](#) in Azure SQL Database servers
- [Managed instances](#)

SQL deployment offers not covered in this guide

- Azure SQL Data Warehouse
- Azure SQL VMs (IaaS)
- SQL Server on-premises

Audience

The intended audiences for this guide are customers facing questions on how to secure Azure SQL Database. The roles interested in this best practice article include, but not limited to:

- Security Architects
- Security Managers
- Compliance Officers
- Privacy Officers
- Security Engineers

Using this guide

This document is intended as a companion to our existing [Azure SQL Database security documentation](#).

Unless otherwise stated, we recommend you follow all best practices listed in each section to achieve the respective goal or requirement. To meet specific security compliance standards or best practices, important regulatory compliance controls are listed under the Requirements or Goals section wherever applicable. These are the security standards and regulations that are referenced in this paper:

- [FedRAMP](#): AC-04, AC-06
- [SOC](#): CM-3, SDL-3
- [ISO/IEC 27001](#): Access Control, Cryptography
- [Microsoft Operational Security Assurance \(OSA\) practices](#): Practice #1-6 and #9

- [NIST Special Publication 800-53 Security Controls](#): AC-5, AC-6
- [PCI DSS](#): 6.3.2, 6.4.2

Feedback

We plan on continuing to update the recommendations and best practices listed here. Provide input or any corrections for this document using the **Feedback** link at the bottom of this article.

Authentication

Authentication is the process of proving the user is who they claim to be. Azure SQL Database supports two types of authentication:

- SQL authentication
- Azure Active Directory authentication

NOTE

Azure Active Directory authentication may not be supported for all tools and 3rd party applications.

Central management for identities

Central identity management offers the following benefits:

- Manage group accounts and control user permissions without duplicating logins across Azure SQL Database servers and databases.
- Simplified and flexible permission management.
- Management of applications at scale.

How to implement:

- Use Azure Active Directory (Azure AD) authentication for centralized identity management.

Best practices:

- Create an Azure AD tenant and [create users](#) to represent human users and create [service principals](#) to represent apps, services, and automation tools. Service principals are equivalent to service accounts in Windows and Linux.
- Assign access rights to resources to Azure AD principals via group assignment: Create Azure AD groups, grant access to groups, and add individual members to the groups. In your database, create contained database users that map your Azure AD groups. To assign permissions inside the database, put users in database roles with the appropriate permissions.
 - See the articles, [Configure and manage Azure Active Directory authentication with SQL](#) and [Use Azure AD for authentication with SQL](#).

NOTE

In a managed instance, you can also create logins that map to Azure AD principals in the master database. See [CREATE LOGIN \(Transact-SQL\)](#).

- Using Azure AD groups simplifies permission management and both the group owner, and the resource owner can add/remove members to/from the group.
- Create a separate group for Azure AD administrator for SQL DB servers.
 - See the article, [Provision an Azure Active Directory administrator for your Azure SQL Database server](#).

- Monitor Azure AD group membership changes using Azure AD audit activity reports.
- For a managed instance, a separate step is required to create Azure AD admin.
 - See the article, [Provision an Azure Active Directory administrator for your managed instance](#).

NOTE

- Azure AD authentication is recorded in Azure SQL audit logs, but not in Azure AD sign-in logs.
- RBAC permissions granted in Azure do not apply to Azure SQL DB permissions. Such permissions must be created/mapped manually in SQL DB using existing SQL permissions.
- On the client-side Azure AD authentication needs access to the internet or via User Defined Route (UDR) to a VNet.
- The Azure AD access token is cached on the client side and its lifetime depends on token configuration. See the article, [Configurable token lifetimes in Azure Active Directory](#)
- For guidance on troubleshooting Azure AD Authentication issues, see the following blog:
<https://techcommunity.microsoft.com/t5/azure-sql-database/troubleshooting-problems-related-to-azure-ad-authentication-with/ba-p/1062991>

Multi-Factor Authentication (MFA)

Mentioned in: OSA Practice #2, ISO Access Control (AC)

Azure Multi-Factor Authentication (MFA) helps provides additional security by requiring more than one form of authentication.

How to implement:

- [Enable MFA](#) in Azure AD using Conditional Access and use interactive authentication.
- The alternative is to enable MFA for the entire Azure AD or AD domain.

Best practices:

- Activate Conditional Access in Azure AD (requires Premium subscription).
 - See the article, [Conditional Access in Azure AD](#).
- Create Azure AD group(s) and enable MFA policy for selected groups using Azure AD Conditional Access.
 - See the article, [Plan Conditional Access Deployment](#).
- MFA can be enabled for the entire Azure AD or for the whole Active Directory federated with Azure AD.
- Use Azure AD Interactive authentication mode for SQL DB where a password is requested interactively, followed by MFA authentication:
 - Use Universal Authentication in SSMS. See the article, [Using Multi-factor AAD authentication with Azure SQL Database and Azure SQL Data Warehouse \(SSMS support for MFA\)](#).
 - Use Interactive Authentication supported in SQL Server Data Tools (SSDT). See the article, [Azure Active Directory support in SQL Server Data Tools \(SSDT\)](#).
 - Use other SQL tools supporting MFA.
 - SSMS Wizard support for export/extract/deploy database
 - [sqlpackage.exe](#): option '/ua'
 - [sqlcmd Utility](#): option -G (interactive)
 - [bcp Utility](#): option -G (interactive)
- Implement your applications to connect to Azure SQL Database using interactive authentication with MFA support.

- See the article, [Connect to Azure SQL Database with Azure Multi-Factor Authentication](#).

NOTE

This authentication mode requires user-based identities. In cases where a trusted identity model is used that is bypassing individual Azure AD user authentication (e.g. using managed identity for Azure resources), MFA does not apply.

Minimize the use of password-based authentication for users

Mentioned in: OSA Practice #4, ISO Access Control (AC)

Password-based authentication methods are a weaker form of authentication. Credentials can be compromised or mistakenly given away.

How to implement:

- Use an Azure AD integrated authentication that eliminates the use of passwords.

Best practices:

- Use single sign-on authentication using Windows credentials. Federate the on-premises AD domain with Azure AD and use Integrated Windows authentication (for domain-joined machines with Azure AD).
 - See the article, [SSMS support for Azure AD Integrated authentication](#).

Minimize the use of password-based authentication for applications

Mentioned in: OSA Practice #4, ISO Access Control (AC)

How to implement:

- Enable Azure Managed Identity. You can also use integrated or certificate-based authentication.

Best practices:

- Use managed identities for Azure resources.
 - [System-assigned managed identity](#)
 - [User-assigned managed identity](#)
 - [Use Azure SQL Database from app service with managed identity \(without code changes\)](#)
- Use cert-based authentication for an application.
 - See this [code sample](#).
- Use Azure AD authentication for integrated federated domain and domain-joined machine (see section above).
 - See the [sample application for integrated authentication](#).

Protect passwords and secrets

For cases when passwords aren't avoidable, make sure they're secured.

How to implement:

- Use Azure Key Vault to store passwords and secrets. Whenever applicable, use MFA for Azure SQL Database with Azure AD users.

Best practices:

- If avoiding passwords or secrets aren't possible, store user passwords and application secrets in Azure Key Vault and manage access through Key Vault access policies.
- Various app development frameworks may also offer framework-specific mechanisms for protecting secrets in the app. For example: [ASP.NET core app](#).

Use SQL authentication for legacy applications

SQL authentication refers to the authentication of a user when connecting to Azure SQL Database using username and password. A login will need to be created in each SQL Database server or a managed instance, and a user created in each database.

How to implement:

- Use SQL authentication.

Best practices:

- As a server admin, create logins and users. Unless using contained database users with passwords, all passwords are stored in master database.
 - See the article, [Controlling and granting database access to SQL Database and SQL Data Warehouse](#).

Access management

Access management is the process of controlling and managing authorized users' access and privileges to Azure SQL Database.

Implement principle of least privilege

Mentioned in: FedRamp controls AC-06, NIST: AC-6, OSA Practice #3

The principle of least privilege states that users shouldn't have more privileges than needed to complete their tasks. For more information, see the article [Just enough administration](#).

How to implement:

Assign only the necessary [permissions](#) to complete the required tasks:

- In SQL Data Plane:
 - Use granular permissions and user-defined database roles (or server-roles in MI):
 1. Create the required roles
 - [CREATE ROLE](#)
 - [CREATE SERVER ROLE](#)
 2. Create required users
 - [CREATE USER](#)
 3. Add users as members to roles
 - [ALTER ROLE](#)
 - [ALTER SERVER ROLE](#)
 4. Then assign permissions to roles.
 - [GRANT](#)
 - Make sure to not assign users to unnecessary roles.
- In Azure Resource Manager:
 - Use builtin-roles if available or custom RBAC roles and assign the necessary permissions.
 - [Built-in roles for Azure](#)

- Custom roles for Azure resources

Best practices:

The following best practices are optional but will result in better manageability and supportability of your security strategy:

- If possible, start with the least possible set of permissions and start adding permissions one by one if there's a real necessity (and justification) – as opposed to the opposite approach: taking permissions away step by step.
- Refrain from assigning permissions to individual users. Use roles (database or server roles) consistently instead. Roles help greatly with reporting and troubleshooting permissions. (Azure RBAC only supports permission assignment via roles.)
- Create and use custom roles with the exact permissions needed. Typical roles that are used in practice:
 - Security deployment
 - Administrator
 - Developer
 - Support personnel
 - Auditor
 - Automated processes
 - End user
- Use built-in roles only when the permissions of the roles match exactly the needed permissions for the user. You can assign users to multiple roles.
- Remember that permissions in SQL Server Database Engine can be applied on the following scopes. The smaller the scope, the smaller the impact of the granted permissions:
 - Azure SQL Database server (special roles in master database)
 - Database
 - Schema
 - It is a best practice to use schemas to grant permissions inside a database. (also see: [Schema-design for SQL Server: recommendations for Schema design with security in mind](#))
 - Object (table, view, procedure, etc.)

NOTE

It is not recommended to apply permissions on the object level because this level adds unnecessary complexity to the overall implementation. If you decide to use object-level permissions, those should be clearly documented. The same applies to column-level-permissions, which are even less recommendable for the same reasons. Also be aware that by default a table-level **DENY** does not override a column-level **GRANT**. This would require the [common criteria compliance Server Configuration](#) to be activated.

- Perform regular checks using [Vulnerability Assessment \(VA\)](#) to test for too many permissions.

Implement Separation of Duties

Mentioned in: FedRamp: AC-04, NIST: AC-5, ISO: A.6.1.2, PCI 6.4.2, SOC: CM-3, SDL-3

Separation of Duties, also called Segregation of Duties describes the requirement to split sensitive tasks into multiple tasks that are assigned to different users. Separation of Duties helps prevent data breaches.

How to implement:

- Identify the required level of Separation of Duties. Examples:
 - Between Development/Test and Production environments
 - Security-wise sensitive tasks vs Database Administrator (DBA) management level tasks vs developer tasks.
 - Examples: Auditor, creation of security policy for Role-level Security (RLS), Implementing SQL Database objects with DDL-permissions.
- Identify a comprehensive hierarchy of users (and automated processes) that access the system.
- Create roles according to the needed user-groups and assign permissions to roles.
 - For management-level tasks in Azure portal or via PowerShell-automation use RBAC roles. Either find a built-in role matching the requirement, or create a custom RBAC role using the available permissions
 - Create Server roles for server-wide tasks (creating new logins, databases) in a managed instance.
 - Create Database Roles for database-level tasks.
- For certain sensitive tasks, consider creating special stored procedures signed by a certificate to execute the tasks on behalf of the users.
 - Example: [Tutorial: Signing Stored Procedures with a Certificate](#)
- Implement Transparent Data Encryption (TDE) with customer-managed keys in Azure Key Vault to enable Separation of Duties between data owner and security owner.
 - See the article, [Configure customer-managed keys for Azure Storage encryption from the Azure portal](#).
- To ensure that a DBA can't see data that is considered highly sensitive and can still do DBA tasks, you can use Always Encrypted with role separation.
 - See the articles, [Overview of Key Management for Always Encrypted](#), [Key Provisioning with Role Separation](#), and [Column Master Key Rotation with Role Separation](#).
- In cases when it isn't feasible at least not without major costs and efforts that may render the system near unusable, compromises can be made and mitigated through the use of compensating controls such as:
 - Human intervention in processes.
 - Audit trails – for more information on Auditing, see, [Audit critical security events](#).

Best practices:

- Make sure that different accounts are used for Development/Test and Production environments. Different accounts help to comply with separation of Test & Production systems.
- Refrain from assigning permissions to individual users. Use roles (database or server roles) consistently instead. Having roles helps greatly with reporting and troubleshooting permissions.
- Use built-in roles when the permissions match exactly the needed permissions – if the union of all permissions from multiple built-in roles leads to a 100% match, you can assign multiple roles concurrently as well.
- Create and use custom roles when built-in roles grant too many permissions or insufficient permissions.
- Role assignments can also be done temporarily, also known as Dynamic Separation of Duties (DSD), either within SQL Agent Job steps in T-SQL or using Azure PIM for RBAC roles.
- Make sure that DBAs don't have access to the encryption keys or key stores and Security Administrators with access to the keys have no access to the database in turn.
- Always make sure to have an Audit trail for security-related actions.
- You can retrieve the definition of the built-in RBAC roles to see the permissions used and create a custom

role based on excerpts and cumulations of these via PowerShell.

- Since any member of the db_owner database role can change security settings like Transparent Data Encryption (TDE), or change the SLO, this membership should be granted with care. However, there are many tasks that require db_owner privileges. Task like changing any database setting such as changing DB options. Auditing plays a key role in any solution.
- It is not possible to restrict permissions of a db_owner, and therefore prevent an administrative account from viewing user data. If there's highly sensitive data in a database, Always Encrypted can be used to safely prevent db_owners or any other DBA from viewing it.

NOTE

Achieving Separation of Duties (SoD) is challenging for security-related or troubleshooting tasks. Other areas like development and end-user roles are easier to segregate. Most compliance related controls allow the use of alternate control functions such as Auditing when other solutions aren't practical.

For the readers that want to dive deeper into SoD, we recommend the following resources:

- For Azure SQL Database:
 - [Controlling and granting database access to SQL Database and SQL Data Warehouse](#)
 - [Engine Separation of Duties for the Application Developer](#)
 - [Separation of Duties in SQL Server 2014](#)
 - [Signing Stored Procedures in SQL Server](#)
- For Azure Resource Management:
 - [Built-in roles for Azure](#)
 - [Custom roles for Azure resources](#)
 - [Using Azure AD Privileged Identity Management for elevated access](#)

Perform regular code reviews

Mentioned in: PCI: 6.3.2, SOC: SDL-3

Separation of Duties is not limited to the data in database, but includes application code. Malicious code can potentially circumvent security controls. Before deploying custom code to production, it is essential to review what's being deployed.

How to implement:

- Use a database tool like Azure Data Studio that supports source control.
- Implement a segregated code deployment process.
- Before committing to main branch, a person (other than the author of the code itself) has to inspect the code for potential elevation of privileges risks as well as malicious data modifications to protect against fraud and rogue access. This can be done using source control mechanisms.

Best practices:

- Standardization: It helps to implement a standard procedure that is to be followed for any code updates.
- Vulnerability Assessment contains rules that check for excessive permissions, the use of old encryption algorithms, and other security problems within a database schema.
- Further checks can be done in a QA or test environment using Advanced Threat Protection that scans for

code that is vulnerable to SQL-injection.

- Examples of what to look out for:
 - Creation of a user or changing security settings from within an automated SQL-code-update deployment.
 - A stored procedure, which, depending on the parameters provided, updates a monetary value in a cell in a non-conforming way.
- Make sure the person conducting the review is an individual other than the originating code author and knowledgeable in code-reviews and secure coding.
- Be sure to know all sources of code-changes. Code can be in T-SQL Scripts. It can be ad-hoc commands to be executed or be deployed in forms of Views, Functions, Triggers, and Stored Procedures. It can be part of SQL Agent Job definitions (Steps). It can also be executed from within SSIS packages, Azure Data Factory, and other services.

Data protection

Data protection is a set of capabilities for safeguarding important information from compromise by encryption or obfuscation.

NOTE

Microsoft attests to Azure SQL Database as being FIPS 140-2 Level 1 compliant. This is done after verifying the strict use of FIPS 140-2 Level 1 acceptable algorithms and FIPS 140-2 Level 1 validated instances of those algorithms including consistency with required key lengths, key management, key generation, and key storage. This attestation is meant to allow our customers to respond to the need or requirement for the use of FIPS 140-2 Level 1 validated instances in the processing of data or delivery of systems or applications. We define the terms "FIPS 140-2 Level 1 compliant" and "FIPS 140-2 Level 1 compliance" used in the above statement to demonstrate their intended applicability to U.S. and Canadian government use of the different term "FIPS 140-2 Level 1 validated."

Encrypt data in transit

Mentioned in: OSA Practice #6, ISO Control Family: Cryptography

Protects your data while data moves between your client and server. Refer to [Network Security](#).

Encrypt data at rest

Mentioned in: OSA Practice #6, ISO Control Family: Cryptography

Encryption at rest is the cryptographic protection of data when it is persisted in database, log, and backup files.

How to implement:

- [Transparent Database Encryption \(TDE\)](#) with service managed keys are enabled by default for any databases created after 2017 in Azure SQL Database.
- In a managed instance, if the database is created from a restore operation using an on-premises server, the TDE setting of the original database will be honored. If the original database doesn't have TDE enabled, we recommend that TDE be manually turned on for the managed instance.

Best practices:

- Don't store data that require encryption-at-rest in the master database. The master database can't be encrypted with TDE.

- Use customer-managed keys in Azure Key Vault if you need increased transparency and granular control over the TDE protection. Azure Key Vault allows the ability to revoke permissions at any time to render the database inaccessible. You can centrally manage TDE protectors along with other keys, or rotate the TDE protector at your own schedule using Azure Key Vault.
- If you're using customer-managed keys in Azure Key Vault, follow the articles, [Guidelines for configuring TDE with Azure Key Vault](#) and [How to configure Geo-DR with Azure Key Vault](#).

Protect sensitive data in use from high-privileged, unauthorized users

Data in use is the data stored in memory of the database system during the execution of SQL queries. If your database stores sensitive data, your organization may be required to ensure that high-privileged users are prevented from viewing sensitive data in your database. High-privilege users, such as Microsoft operators or DBAs in your organization should be able to manage the database, but prevented from viewing and potentially exfiltrating sensitive data from the memory of the SQL Server process or by querying the database.

The policies that determine which data is sensitive and whether the sensitive data must be encrypted in memory and not accessible to administrators in plaintext, are specific to your organization and compliance regulations you need to adhere to. Please see the related requirement: [Identify and tag sensitive data](#).

How to implement:

- Use [Always Encrypted](#) to ensure sensitive data isn't exposed in plaintext in Azure SQL Database, even in memory/in use. Always Encrypted protects the data from Database Administrators (DBAs) and cloud admins (or bad actors who can impersonate high-privileged but unauthorized users) and gives you more control over who can access your data.

Best practices:

- Always Encrypted isn't a substitute to encrypt data at rest (TDE) or in transit (SSL/TLS). Always Encrypted shouldn't be used for non-sensitive data to minimize performance and functionality impact. Using Always Encrypted in conjunction with TDE and Transport Layer Security (TLS) is recommended for comprehensive protection of data at-rest, in-transit, and in-use.
- Assess the impact of encrypting the identified sensitive data columns before you deploy Always Encrypted in a production database. In general, Always Encrypted reduces the functionality of queries on encrypted columns and has other limitations, listed in [Always Encrypted - Feature Details](#). Therefore, you may need to rearchitect your application to re-implement the functionality, a query does not support, on the client side or/and refactor your database schema, including the definitions of stored procedures, functions, views and triggers. Existing applications may not work with encrypted columns if they do not adhere to the restrictions and limitations of Always Encrypted. While the ecosystem of Microsoft tools, products and services supporting Always Encrypted is growing, a number of them do not work with encrypted columns. Encrypting a column may also impact query performance, depending on the characteristics of your workload.
- Manage Always Encrypted keys with role separation if you're using Always Encrypted to protect data from malicious DBAs. With role separation, a security admin creates the physical keys. The DBA creates the column master key and column encryption key metadata objects, describing the physical keys, in the database. During this process, the security admin doesn't need access to the database, and the DBA doesn't need access to the physical keys in plaintext.
 - See the article, [Managing Keys with Role Separation](#) for details.
- Store your column master keys in Azure Key Vault for ease of management. Avoid using Windows Certificate Store (and in general, distributed key store solutions, as opposed central key management solutions) that make key management hard.
- Think carefully through the tradeoffs of using multiple keys (column master key or column encryption keys).

Keep the number of keys small to reduce key management cost. One column master key and one column encryption key per database is typically sufficient in steady-state environments (not in the middle of a key rotation). You may need additional keys if you have different user groups, each using different keys and accessing different data.

- Rotate column master keys per your compliance requirements. If you also need to rotate column encryption keys, consider using online encryption to minimize application downtime.
 - See the article, [Performance and Availability Considerations](#).
- Use deterministic encryption if computations (equality) on data need to be supported. Otherwise, use randomized encryption. Avoid using deterministic encryption for low-entropy data sets, or data sets with publicly known distribution.
- If you're concerned about third-party accessing your data legally without your consent, ensure that all application and tools that have access to the keys and data in plaintext run outside of Microsoft Azure Cloud. Without access to the keys, the third party will have no way of decrypting the data unless they bypass the encryption.
- Always Encrypted doesn't easily support granting temporary access to the keys (and the protected data). For example, if you need to share the keys with a DBA to allow the DBA to do some cleansing operations on sensitive and encrypted data. The only way to reliably revoke the access to the data from the DBA will be to rotate both the column encryption keys and the column master keys protecting the data, which is an expensive operation.
- To access the plaintext values in encrypted columns, a user needs to have access to the CMK that protects columns, which is configured in the key store holding the CMK. The user also needs to have the **VIEW ANY COLUMN MASTER KEY DEFINITION** and **VIEW ANY COLUMN ENCRYPTION KEY DEFINITION** database permissions.

Control access of application users to sensitive data through encryption

Encryption can be used as a way to ensure that only specific application users who have access to cryptographic keys can view or update the data.

How to implement:

- Use Cell-level Encryption (CLE). See the article, [Encrypt a Column of Data](#) for details.
- Use Always Encrypted, but be aware of its limitation. The limitations are listed below.

Best practices

When using CLE:

- Control access to keys through SQL permissions and roles.
- Use AES (AES 256 recommended) for data encryption. Algorithms, such RC4, DES and TripleDES, are deprecated and shouldn't be used because of known vulnerabilities.
- Protect symmetric keys with asymmetric keys/certificates (not passwords) to avoid using 3DES.
- Be careful when migrating a database using Cell-Level Encryption via export/import (bacpac files).
 - See the article, [Recommendations for using Cell Level Encryption in Azure SQL Database](#) on how to prevent losing keys when migrating data, and for other best practice guidance.

Keep in mind that Always Encrypted is primarily designed to protect sensitive data in use from high-privilege users of Azure SQL Database (cloud operators, DBAs) - see [Protect sensitive data in use from high-privileged, unauthorized users](#). Be aware of the following challenges when using Always Encrypted to protect data from application users:

- By default, all Microsoft client drivers supporting Always Encrypted maintain a global (one per application) cache of column encryption keys. Once a client driver acquires a plaintext column encryption key by contacting a key store holding a column master key, the plaintext column encryption key is cached. This makes isolating data from users of a multi-user application challenging. If your application impersonates end users when interacting with a key store (such as Azure Key Vault), after a user's query populates the cache with a column encryption key, a subsequent query that requires the same key but is triggered by another user will use the cached key. The driver won't call the key store and it won't check if the second user has a permission to access the column encryption key. As a result, the user will see the encrypted data even if the user doesn't have access to the keys. To achieve the isolation of users within a multi-user application, you can disable column encryption key caching. Disabling caching will cause additional performance overheads, as the driver will need to contact the key store for each data encryption or decryption operation.

Protect data against unauthorized viewing by application users while preserving data format

Another technique for preventing unauthorized users from viewing data is to obfuscate or mask the data while preserving data types and formats to ensure that user applications can continue handle and display the data.

How to implement:

- Use [Dynamic Data Masking](#) to obfuscate table columns.

NOTE

Always Encrypted does not work with Dynamic Data Masking. It is not possible to encrypt and mask the same column, which implies that you need to prioritize protecting data in use vs. masking the data for your app users via Dynamic Data Masking.

Best practices:

NOTE

Dynamic Data Masking cannot be used to protect data from high-privilege users. Masking policies do not apply to users with administrative access like db_owner.

- Don't permit app users to run ad-hoc queries (as they may be able to work around Dynamic Data Masking).
 - See the article, [Bypassing masking using inference or brute-force techniques](#) for details.
- Use a proper access control policy (via SQL permissions, roles, RLS) to limit user permissions to make updates in the masked columns. Creating a mask on a column doesn't prevent updates to that column. Users that receive masked data when querying the masked column, can update the data if they have write-permissions.
- Dynamic Data Masking doesn't preserve the statistical properties of the masked values. This may impact query results (for example, queries containing filtering predicates or joins on the masked data).

Network security

Network security refers to access controls and best practices to secure your data in transit to Azure SQL Database.

Configure my client to connect securely to Azure SQL Database

Best practices on how to prevent client machines and applications with well-known vulnerabilities (for example, using older TLS protocols and cipher suites) from connecting to Azure SQL Database.

How to implement:

- Ensure that client machines connecting to Azure SQL Database are using [Transport Layer Security \(TLS\)](#).

Best practices:

- Configure all your apps and tools to connect to SQL DB with encryption enabled
 - Encrypt = On, TrustServerCertificate = Off (or equivalent with non-Microsoft drivers).
- If your app uses a driver that doesn't support TLS or supports an older version of TLS, replace the driver, if possible. If not possible, carefully evaluate the security risks.
- Reduce attack vectors via vulnerabilities in SSL 2.0, SSL 3.0, TLS 1.0, and TLS 1.1 by disabling them on client machines connecting to Azure SQL Database per [Transport Layer Security \(TLS\) registry settings](#).
- Check cipher suites available on the client: [Cipher Suites in TLS/SSL \(Schannel SSP\)](#). Specifically, disable 3DES per [Configuring TLS Cipher Suite Order](#).
- For Azure SQL Database, encryption is enforced for both Proxy and Redirect connection types. If you're using a managed instance, use the **Proxy** connection type (default) as this enforces encryption from the server side. The **Redirect** connection type currently doesn't support encryption enforcement and is only available on private IP connections.
- For more information, see [Azure SQL Connectivity Architecture - Connection policy](#).

Minimize Attack Surface

Minimize the number of features that can be attacked by a malicious user. Implement network access controls for Azure SQL Database.

Mentioned in: OSA Practice #5

How to implement:

In an Azure SQL Database server (containing singleton database or elastic pools):

- Set Allow Access to Azure services to OFF.
- Use VNet Service endpoints and VNet Firewall Rules.
- Use Private Link (preview).

In a managed instance:

- Follow the guidelines in [Network requirements](#).

Best practices:

- Restricting access to Azure SQL Database by connecting on a private endpoint (for example, using a private data path):
 - A managed instance can be isolated inside a VNet to prevent external access. Applications and tools that are in the same or peered VNet in the same region could access it directly. Applications and tools that are in different region could use VNet-to-VNet connection or ExpressRoute circuit peering to establish connection. Customer should use Network Security Groups (NSG) to restrict access over port 1433 only to resources that require access to a managed instance
 - For a SQL Database server (containing single databases or elastic pools), use the [Private Link](#) feature that provides a dedicated private IP for the SQL Database server inside your VNet. You can also use [VNet Service endpoints with VNet Firewall Rules](#) to restrict access to your SQL Database servers.
 - Mobile users should use point-to-site VPN connections to connect over the data path.
 - Users connected to their on-premises network should use site-to-site VPN connection or ExpressRoute to connect over the data path.
- You can access Azure SQL Database by connecting to a public endpoint (for example, using a public data

path). The following best practices should be considered:

- For a SQL Database server, use [IP firewall rules](#) to restrict access to only authorized IP addresses.
- For a managed instance, use Network Security Groups (NSG) to restrict access over port 3342 only to required resources. For more information, see [Use an Azure SQL Database managed instance securely with public endpoints](#).

NOTE

A managed instance public endpoint is not enabled by default and it must be explicitly enabled. If company policy disallows the use of public endpoints, use [Azure Policy](#) to prevent enabling public endpoints in the first place.

- Set up Azure Networking components:
 - Follow [Azure best practices for network security](#).
 - Plan Virtual Network (VNet) configuration per best practices outlined in [Azure Virtual Network frequently asked questions \(FAQ\)](#) and plan.
 - Segment a VNet into multiple subnets and assign resources for similar role to the same subnet (for example, front-end vs back-end resources).
 - Use [Network Security Groups \(NSGs\)](#) to control traffic between subnets inside the Azure VNet boundary.
 - Enable [Azure Network Watcher](#) for your subscription to monitor inbound and outbound network traffic.

Configure Power BI for secure connections to Azure SQL Database

Best practices:

- For Power BI Desktop, use private data path whenever possible.
- Ensure that Power BI Desktop is connecting using TLS1.2 by setting the registry key on the client machine as per [Transport Layer Security \(TLS\)](#) registry settings.
- Restrict data access for specific users via [Row-level security \(RLS\) with Power BI](#).
- For Power BI Service, use the [on-premises data gateway](#), keeping in mind [Limitations and Considerations](#).

Configure App Service for secure connections to Azure SQL Database

Best practices:

- For a simple Web App, connecting over public endpoint requires setting **Allow Azure Services** to ON.
- [Integrate your app with an Azure Virtual Network](#) for private data path connectivity to a managed instance. Optionally, you can also deploy a Web App with [App Service Environments \(ASE\)](#).
- For Web App with ASE or VNet Integrated Web App connecting to a database in SQL Database server, you can use [VNet Service endpoints and VNet Firewall Rules](#) to limit access from a specific VNet and subnet. Then set **Allow Azure Services** to OFF. You can also connect ASE to a managed instance over a private data path.
- Ensure that your Web App is configured per the article, [Best practices for securing PaaS web and mobile applications using Azure App Service](#).
- Install [Web Application Firewall \(WAF\)](#) to protect your web app from common exploits and vulnerabilities.

Configure Azure VM hosting for secure connections to Azure SQL Database

Best practices:

- Use a combination of Allow and Deny rules on the NSGs of Azure VMs to control which regions can be accessed from the VM.

- Ensure that your VM is configured per the article, [Security best practices for IaaS workloads in Azure](#).
- Ensure that all VMs are associated with a specific VNet and subnet.
- Evaluate if you need the default route 0.0.0.0/Internet per the guidance at [about forced tunneling](#).
 - If yes – for example, front-end subnet - then keep the default route.
 - If no – for example, middle tier or back-end subnet – then enable force tunneling so no traffic goes over Internet to reach on-premises (a.k.a cross-premises).
- Implement [optional default routes](#) if you're using peering or connecting to on-premises.
- Implement [User Defined Routes](#) if you need to send all traffic in the VNet to a Network Virtual Appliance for packet inspection.
- Use [VNet Service endpoints](#) for secure access to PaaS services like Azure Storage via the Azure backbone network.

Protect against Distributed Denial of Service (DDoS) attacks

Distributed Denial of Service (DDoS) attacks are attempts by a malicious user to send a flood of network traffic to Azure SQL Database with the aim of overwhelming the Azure infrastructure and causing it to reject valid logins and workload.

Mentioned in: OSA Practice #9

How to implement:

DDoS protection is automatically enabled as part of the Azure Platform. It includes always-on traffic monitoring and real-time mitigation of network-level attacks on public endpoints.

- Use [Azure DDoS Protection](#) to monitor public IP addresses associated to resources deployed in VNets.
- Use [Advanced Threat Protection for Azure SQL Database](#) to detect Denial of Service (DoS) attacks against databases.

Best practices:

- Follow the practices described in [Minimize Attack Surface](#) helps minimize DDoS attack threats.
- The Advanced Threat Protection **Brute force SQL credentials** alert helps to detect brute force attacks. In some cases, the alert can even distinguish penetration testing workloads.
- For Azure VM hosting applications connecting to SQL Database:
 - Follow recommendation to Restrict access through Internet-facing endpoints in Azure Security Center.
 - Use virtual machine scale sets to run multiple instances of your application on Azure VMs.
 - Disable RDP and SSH from Internet to prevent brute force attack.

Monitoring, Logging, and Auditing

This section refers to capabilities to help you detect anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases. It also describes best practices to configure database auditing to track and capture database events.

Protect databases against attacks

Advanced threat protection enables you to detect and respond to potential threats as they occur by providing security alerts on anomalous activities.

How to implement:

- Use [Advanced Threat Protection for SQL](#) to detect unusual and potentially harmful attempts to access or exploit databases, including:
 - SQL injection attack.
 - Credentials theft/leak.
 - Privilege abuse.
 - Data exfiltration.

Best practices:

- Configure [Advanced Data Security \(ADS\)](#) for Azure SQL Database for a specific SQL Database server or a managed instance. You can also configure ADS for all SQL Database servers and managed instances in a subscription by switching to [Azure Security Center Standard tier](#).
- For a full investigation experience, it's recommended to enable [SQL Database Auditing](#). With auditing, you can track database events and write them to an audit log in an Azure Storage account or Azure Log Analytics workspace.

Audit critical security events

Tracking of database events helps you understand database activity. You can gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations. It also enables and facilitates adherence to compliance standards.

How to implement:

- Enable [SQL Database Auditing](#) to track database events and write them to an audit log in your Azure Storage account, Log Analytics workspace (preview), or Event Hubs (preview).
- Audit logs can be written to an Azure Storage account, to a Log Analytics workspace for consumption by Azure Monitor logs, or to event hub for consumption using event hub. You can configure any combination of these options, and audit logs will be written to each.

Best practices:

- By configuring [SQL Database Auditing](#) on your database server to audit events, all existing and newly created databases on that server will be audited.
- By default auditing policy includes all actions (queries, stored procedures and successful and failed logins) against the databases, which may result in high volume of audit logs. It's recommended for customers to [configure auditing for different types of actions and action groups using PowerShell](#). Configuring this will help control the number of audited actions, and minimize the risk of event loss. Custom audit configuration allow customers to capture only the audit data that is needed.
- Audit logs can be consumed directly in the [Azure portal](#), or from the storage location that was configured.

NOTE

Enabling auditing to Log Analytics will incur cost based on ingestion rates. Please be aware of the associated cost with using this [option](#), or consider storing the audit logs in an Azure storage account.

Further resources:

- [SQL Database Auditing](#)
- [SQL Server Auditing](#)

Secure audit logs

Restrict access to the storage account to support Separation of Duties and to separate DBA from Auditors.

How to implement:

- When saving Audit logs to Azure Storage, make sure that access to the Storage Account is restricted to the minimal security principles. Control who has access to the storage account.
 - For more information, see [Authorizing access to Azure Storage](#).

Best practices:

- Controlling Access to the Audit Target is a key concept in separating DBA from Auditors.
- When auditing access to sensitive data, consider securing the data with data encryption to avoid information leakage to the Auditor. For more information, see the section [Protect sensitive data in use from high-privileged, unauthorized users](#).

Security Management

This section describes the different aspects and best practices for managing your databases security posture. It includes best practices for ensuring your databases are configured to meet security standards, for discovering and for classifying and tracking access to potentially sensitive data in your databases.

Ensure that the database(s) are configured to meet security best practices

Proactively improve your database security by discovering and remediating potential database vulnerabilities.

How to implement:

- Enable [SQL Vulnerability Assessment](#) (VA) to scan your database for security issues, and to automatically run periodically on your databases.

Best practices:

- Initially, run VA on your databases and iterate by remediating failing checks that oppose security best practices. Set up baselines for acceptable configurations until the scan comes out *clean*, or all checks have passed.
- Configure periodic recurring scans to run once a week and configure the relevant person to receive summary emails.
- Review the VA summary following each weekly scan. For any vulnerabilities found, evaluate the drift from the previous scan result and determine if the check should be resolved. Review if there's a legitimate reason for the change in configuration.
- Resolve checks and update baselines where relevant. Create ticket items for resolving actions and track these until they're resolved.

Further resources:

- [SQL Vulnerability Assessment](#)
- [SQL Vulnerability Assessment service helps you identify database vulnerabilities](#)

Identify and tag sensitive data

Discover columns that potentially contain sensitive data. What is considered sensitive data heavily depends on the customer, compliance regulation, etc., and needs to be evaluated by the users in charge of that data. Classify the columns to use advanced sensitivity-based auditing and protection scenarios.

How to implement:

- Use [SQL Data Discovery and Classification](#) to discover, classify, label, and protect the sensitive data in your databases.
 - View the classification recommendations that are created by the automated discovery in the SQL Data Discovery and Classification dashboard. Accept the relevant classifications, such that your sensitive data

- is persistently tagged with classification labels.
- Manually add classifications for any additional sensitive data fields that were not discovered by the automated mechanism.
 - For more information, see [SQL Data Discovery & Classification](#).

Best practices:

- Monitor the classification dashboard on a regular basis for an accurate assessment of the database's classification state. A report on the database classification state can be exported or printed to share for compliance and auditing purposes.
- Continuously monitor the status of recommended sensitive data in SQL Vulnerability Assessment. Track the sensitive data discovery rule and identify any drift in the recommended columns for classification.
- Use classification in a way that is tailored to the specific needs of your organization. Customize your Information Protection policy (sensitivity labels, information types, discovery logic) in the [SQL Information Protection](#) policy in Azure Security Center.

Track access to sensitive data

Monitor who accesses sensitive data and capture queries on sensitive data in audit logs.

How to implement:

- Use SQL Audit and Data Classification in combination.
 - In your [SQL Database Audit](#) log, you can track access specifically to sensitive data. You can also view information such as the data that was accessed, as well as its sensitivity label. For more information, see [Auditing access to sensitive data](#).

Best practices:

- See best practices for the Auditing and Data Classification sections:
 - [Audit critical security events](#)
 - [Identify and tag sensitive data](#)

Visualize security and compliance status

Use a unified infrastructure security management system that strengthens the security posture of your data centers (including SQL Databases). View a list of recommendations concerning the security of your databases and compliance status.

How to implement:

- Monitor SQL-related security recommendations and active threats in [Azure Security Center](#).

Common security threats and potential mitigations

This section helps you find security measures to protect against certain attack vectors. It's expected that most mitigations can be achieved by following one or more of the security guidelines above.

Security threat: Data exfiltration

Data exfiltration is the unauthorized copying, transfer, or retrieval of data from a computer or server. See a definition for [data exfiltration](#) on Wikipedia.

Connecting to Azure SQL Database server over a public endpoint presents a data exfiltration risk as it requires customers to open their firewalls to public IPs.

Scenario 1: An application on an Azure VM connects to a database in an Azure SQL Database server. A rogue actor gets access to the VM and compromises it. In this scenario, data exfiltration means that an external entity

using the rogue VM connects to the database, copies personal data, and stores it in a blob storage or a different SQL Database in a different subscription.

Scenario 2: A Rouge DBA. This scenario is often raised by security sensitive customers from regulated industries. In this scenario, a high privilege user might copy data from Azure SQL Database to another subscription not controlled by the data owner.

Potential mitigations:

Today, Azure SQL Database offers the following techniques for mitigating data exfiltration threats:

- Use a combination of Allow and Deny rules on the NSGs of Azure VMs to control which regions can be accessed from the VM.
- If using an Azure SQL Database server (containing singleton databases or elastic pools), set the following options:
 - Allow Azure Services to OFF.
 - Only allow traffic from the subnet containing your Azure VM by setting up a VNet Firewall rule.
 - Use [Private Link](#)
- For a managed instance, using private IP access by default addresses the first data exfiltration concern of a rogue VM. Turn on the subnet delegation feature on a subnet to automatically set the most restrictive policy on a managed instance subnet.
- The Rogue DBA concern is more exposed with a managed instance as it has a larger surface area and networking requirements are visible to customers. The best mitigation for this is applying all of the practices in this security guide to prevent the Rogue DBA scenario in the first place (not only for data exfiltration). Always Encrypted is one method to protect sensitive data by encrypting it and keeping the key inaccessible for the DBA.

Security aspects of business continuity and availability

Most security standards address data availability in terms of operational continuity, achieved by implementing redundancy and fail-over capabilities to avoid single points of failure. For disaster scenarios, it's a common practice to keep backups of Data and Log files. The following section provides a high-level overview of the capabilities that are built-into Azure. It also provides additional options that can be configured to meet specific needs:

- Azure offers built-in high-availability: [High-availability and Azure SQL Database](#)
- The Business Critical tier includes failover groups, multi-availability zones, full and differential log backups, and point-in-time-restore backups enabled by default:
 - [High-availability and Azure SQL Database - Zone redundant configuration](#)
 - [Automated backups](#)
 - [Recover an Azure SQL Database using automated database backups - Point-in-time restore](#)
- Additional business continuity features such as auto-failover groups across different Azure geos can be configured as described here: [Overview of business continuity with Azure SQL Database](#)

Next steps

- See [An overview of Azure SQL Database security capabilities](#)

Azure SQL Database and SQL Data Warehouse access control

11/7/2019 • 4 minutes to read • [Edit Online](#)

To provide security, Azure [SQL Database](#) and [SQL Data Warehouse](#) control access with firewall rules limiting connectivity by IP address, authentication mechanisms requiring users to prove their identity, and authorization mechanisms limiting users to specific actions and data.

IMPORTANT

For an overview of the SQL Database security features, see [SQL security overview](#). For a tutorial, see [Secure your Azure SQL Database](#). For an overview of SQL Data Warehouse security features, see [SQL Data Warehouse security overview](#)

Firewall and firewall rules

Microsoft Azure SQL Database provides a relational database service for Azure and other Internet-based applications. To help protect your data, firewalls prevent all access to your database server until you specify which computers have permission. The firewall grants access to databases based on the originating IP address of each request. For more information, see [Overview of Azure SQL Database firewall rules](#)

The Azure SQL Database service is only available through TCP port 1433. To access a SQL Database from your computer, ensure that your client computer firewall allows outgoing TCP communication on TCP port 1433. If not needed for other applications, block inbound connections on TCP port 1433.

As part of the connection process, connections from Azure virtual machines are redirected to a different IP address and port, unique for each worker role. The port number is in the range from 11000 to 11999. For more information about TCP ports, see [Ports beyond 1433 for ADO.NET 4.5 and SQL Database2](#).

Authentication

SQL Database supports two types of authentication:

- **SQL Authentication:**

This authentication method uses a username and password. When you created the SQL Database server for your database, you specified a "server admin" login with a username and password. Using these credentials, you can authenticate to any database on that server as the database owner, or "dbo."

- **Azure Active Directory Authentication:**

This authentication method uses identities managed by Azure Active Directory and is supported for managed and integrated domains. If you want to use Azure Active Directory Authentication, you must create another server admin called the "Azure AD admin," which is allowed to administer Azure AD users and groups. This admin can also perform all operations that a regular server admin can. See [Connecting to SQL Database By Using Azure Active Directory Authentication](#) for a walkthrough of how to create an Azure AD admin to enable Azure Active Directory Authentication.

The Database Engine closes connections that remain idle for more than 30 minutes. The connection must login again before it can be used. Continuously active connections to SQL Database require reauthorization (performed by the database engine) at least every 10 hours. The database engine attempts reauthorization using the originally submitted password and no user input is required. For performance reasons, when a password is reset in SQL

Database, the connection is not reauthenticated, even if the connection is reset due to connection pooling. This is different from the behavior of on-premises SQL Server. If the password has been changed since the connection was initially authorized, the connection must be terminated and a new connection made using the new password. A user with the `KILL DATABASE CONNECTION` permission can explicitly terminate a connection to SQL Database by using the `KILL` command.

User accounts can be created in the master database and can be granted permissions in all databases on the server, or they can be created in the database itself (called contained users). For information on creating and managing logins, see [Manage logins](#). Use contained databases to enhance portability and scalability. For more information on contained users, see [Contained Database Users - Making Your Database Portable](#), [CREATE USER \(Transact-SQL\)](#), and [Contained Databases](#).

As a best practice your application should use a dedicated account to authenticate -- this way you can limit the permissions granted to the application and reduce the risks of malicious activity in case your application code is vulnerable to a SQL injection attack. The recommended approach is to create a [contained database user](#), which allows your app to authenticate directly to the database.

Authorization

Authorization refers to what a user can do within an Azure SQL Database, and this is controlled by your user account's database [role memberships](#) and [object-level permissions](#). As a best practice, you should grant users the least privileges necessary. The server admin account you are connecting with is a member of `db_owner`, which has authority to do anything within the database. Save this account for deploying schema upgrades and other management operations. Use the "ApplicationUser" account with more limited permissions to connect from your application to the database with the least privileges needed by your application. For more information, see [Manage logins](#).

Typically, only administrators need access to the `master` database. Routine access to each user database should be through non-administrator contained database users created in each database. When you use contained database users, you do not need to create logins in the `master` database. For more information, see [Contained Database Users - Making Your Database Portable](#).

You should familiarize yourself with the following features that can be used to limit or elevate permissions:

- [Impersonation](#) and [module-signing](#) can be used to securely elevate permissions temporarily.
- [Row-Level Security](#) can be used limit which rows a user can access.
- [Data Masking](#) can be used to limit exposure of sensitive data.
- [Stored procedures](#) can be used to limit the actions that can be taken on the database.

Next steps

- For an overview of the SQL Database security features, see [SQL security overview](#).
- To learn more about firewall rules, see [Firewall rules](#).
- To learn about users and logins, see [Manage logins](#).
- For a discussion of proactive monitoring, see [Database Auditing](#) and [SQL Database threat detection](#).
- For a tutorial, see [Secure your Azure SQL Database](#).

Controlling and granting database access to SQL Database and SQL Data Warehouse

11/18/2019 • 12 minutes to read • [Edit Online](#)

After firewall rules configuration, you can connect to Azure [SQL Database](#) and [SQL Data Warehouse](#) as one of the administrator accounts, as the database owner, or as a database user in the database.

NOTE

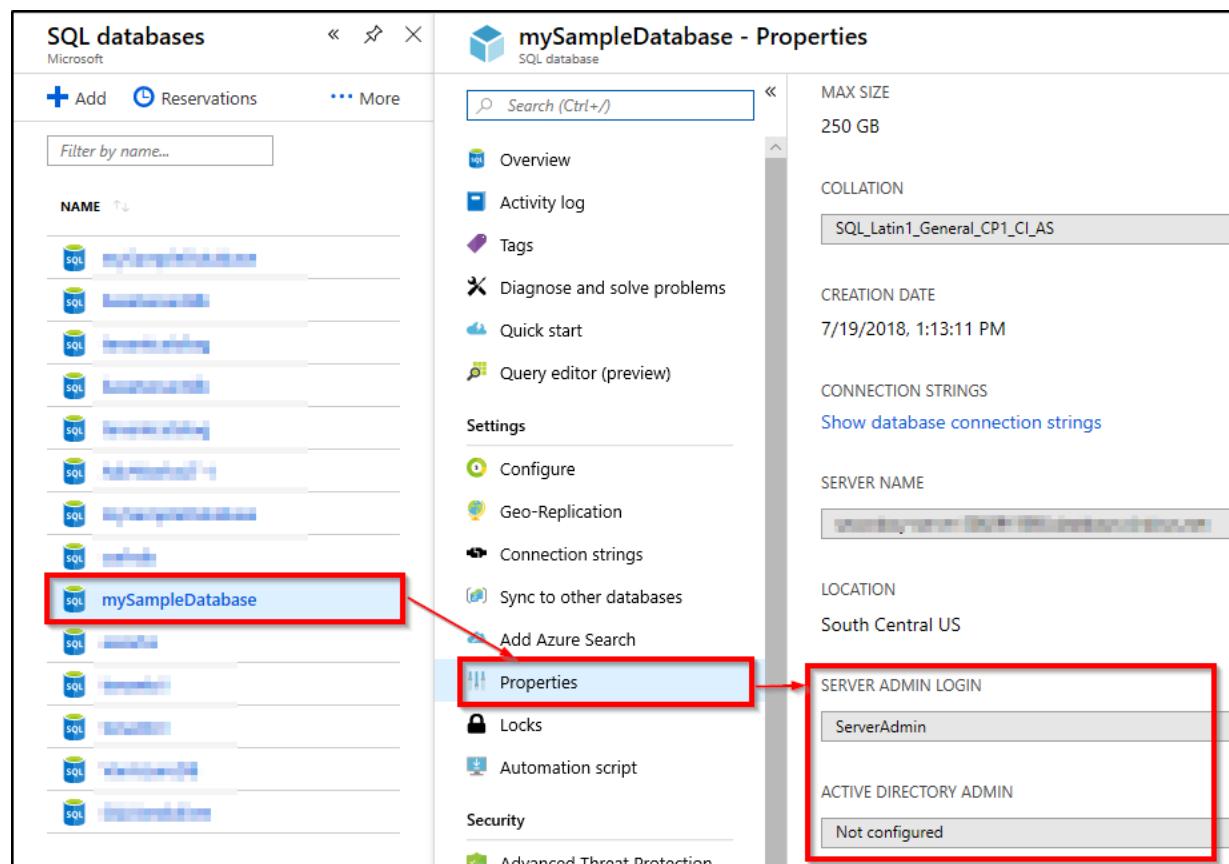
This topic applies to Azure SQL server, and to SQL Database and SQL Data Warehouse databases created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

TIP

For a tutorial, see [Secure your Azure SQL Database](#). This tutorial does not apply to [Azure SQL Database Managed Instance](#).

Unrestricted administrative accounts

There are two administrative accounts (**Server admin** and **Active Directory admin**) that act as administrators. To identify these administrator accounts for your SQL server, open the Azure portal, and navigate to the Properties tab of your SQL server or SQL Database.



- **Server admin**

When you create an Azure SQL server, you must designate a **Server admin login**. SQL server creates that account as a login in the master database. This account connects using SQL Server authentication (user name and password). Only one of these accounts can exist.

NOTE

To reset the password for the server admin, go to the [Azure portal](#), click **SQL Servers**, select the server from the list, and then click **Reset Password**.

- **Azure Active Directory admin**

One Azure Active Directory account, either an individual or security group account, can also be configured as an administrator. It is optional to configure an Azure AD administrator, but an Azure AD administrator **must** be configured if you want to use Azure AD accounts to connect to SQL Database. For more information about configuring Azure Active Directory access, see [Connecting to SQL Database or SQL Data Warehouse By Using Azure Active Directory Authentication](#) and [SSMS support for Azure AD MFA with SQL Database and SQL Data Warehouse](#).

The **Server admin** and **Azure AD admin** accounts have the following characteristics:

- Are the only accounts that can automatically connect to any SQL Database on the server. (To connect to a user database, other accounts must either be the owner of the database, or have a user account in the user database.)
- These accounts enter user databases as the `dbo` user and they have all the permissions in the user databases. (The owner of a user database also enters the database as the `dbo` user.)
- Do not enter the `master` database as the `dbo` user, and have limited permissions in master.
- Are **not** members of the standard SQL Server `sysadmin` fixed server role, which is not available in SQL database.
- Can create, alter, and drop databases, logins, users in master, and server-level IP firewall rules.
- Can add and remove members to the `dbmanager` and `loginmanager` roles.
- Can view the `sys.sql_logins` system table.
- Cannot be renamed.
- To change the Azure AD admin account, use the Portal or Azure CLI.
- The Server Admin account cannot be changed afterwards.

Configuring the firewall

When the server-level firewall is configured for an individual IP address or range, the **SQL server admin** and the **Azure Active Directory admin** can connect to the master database and all the user databases. The initial server-level firewall can be configured through the [Azure portal](#), using [PowerShell](#) or using the [REST API](#). Once a connection is made, additional server-level IP firewall rules can also be configured by using [Transact-SQL](#).

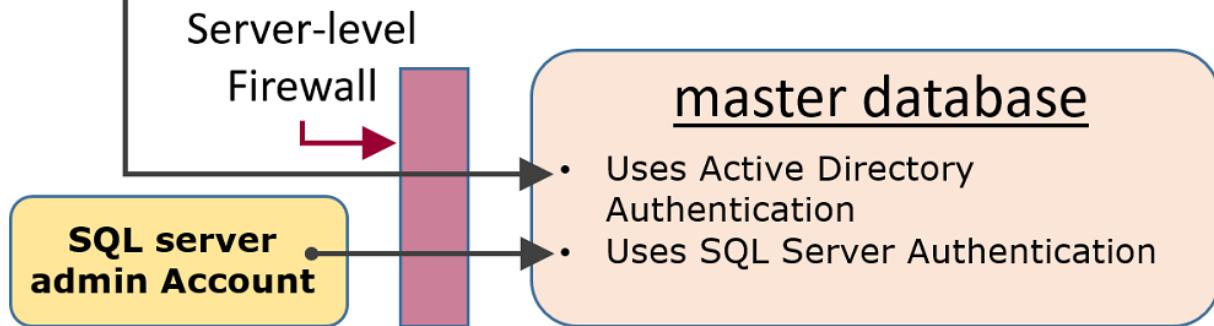
Administrator access path

When the server-level firewall is properly configured, the **SQL server admin** and the **Azure Active Directory admin** can connect using client tools such as SQL Server Management Studio or SQL Server Data Tools. Only the latest tools provide all the features and capabilities. The following diagram shows a typical configuration for the two administrator accounts.

Azure Active Directory

Azure Active Directory admin

- A single account (can be a user or group)
- Optional if not using Azure Active Directory Authentication
- Required if using Azure Active Directory Authentication



When using an open port in the server-level firewall, administrators can connect to any SQL Database.

Connecting to a database by using SQL Server Management Studio

For a walk-through of creating a server, a database, server-level IP firewall rules, and using SQL Server Management Studio to query a database, see [Get started with Azure SQL Database servers, databases, and firewall rules by using the Azure portal and SQL Server Management Studio](#).

IMPORTANT

It is recommended that you always use the latest version of Management Studio to remain synchronized with updates to Microsoft Azure and SQL Database. [Update SQL Server Management Studio](#).

Additional server-level administrative roles

IMPORTANT

This section does not apply to **Azure SQL Database Managed Instance** as these roles are specific to **Azure SQL Database**.

In addition to the server-level administrative roles discussed previously, SQL Database provides two restricted administrative roles in the master database to which user accounts can be added that grant permissions to either create databases or manage logins.

Database creators

One of these administrative roles is the **dbmanager** role. Members of this role can create new databases. To use this role, you create a user in the `master` database and then add the user to the **dbmanager** database role. To create a database, the user must be a user based on a SQL Server login in the `master` database or contained database user based on an Azure Active Directory user.

1. Using an administrator account, connect to the `master` database.
2. Create a SQL Server authentication login, using the `CREATE LOGIN` statement. Sample statement:

```
CREATE LOGIN Mary WITH PASSWORD = '<strong_password>';
```

NOTE

Use a strong password when creating a login or contained database user. For more information, see [Strong Passwords](#).

To improve performance, logins (server-level principals) are temporarily cached at the database level. To refresh the authentication cache, see [DBCC FLUSHAUTHCACHE](#).

3. In the `master` database, create a user by using the [CREATE USER](#) statement. The user can be an Azure Active Directory authentication contained database user (if you have configured your environment for Azure AD authentication), or a SQL Server authentication contained database user, or a SQL Server authentication user based on a SQL Server authentication login (created in the previous step.) Sample statements:

```
CREATE USER [mike@contoso.com] FROM EXTERNAL PROVIDER; -- To create a user with Azure Active  
Directory  
CREATE USER Ann WITH PASSWORD = '<strong_password>'; -- To create a SQL Database contained database  
user  
CREATE USER Mary FROM LOGIN Mary; -- To create a SQL Server user based on a SQL Server  
authentication login
```

4. Add the new user, to the **dbmanager** database role in `master` using the [ALTER ROLE](#) statement.
Sample statements:

```
ALTER ROLE dbmanager ADD MEMBER Mary;  
ALTER ROLE dbmanager ADD MEMBER [mike@contoso.com];
```

NOTE

The dbmanager is a database role in master database so you can only add a database user to the dbmanager role. You cannot add a server-level login to database-level role.

5. If necessary, configure a firewall rule to allow the new user to connect. (The new user might be covered by an existing firewall rule.)

Now the user can connect to the `master` database and can create new databases. The account creating the database becomes the owner of the database.

Login managers

The other administrative role is the login manager role. Members of this role can create new logins in the master database. If you wish, you can complete the same steps (create a login and user, and add a user to the **loginmanager** role) to enable a user to create new logins in the master. Usually logins are not necessary as Microsoft recommends using contained database users, which authenticate at the database-level instead of using users based on logins. For more information, see [Contained Database Users - Making Your Database Portable](#).

Non-administrator users

Generally, non-administrator accounts do not need access to the master database. Create contained database users at the database level using the [CREATE USER \(Transact-SQL\)](#) statement. The user can be an Azure Active Directory authentication contained database user (if you have configured your environment for Azure AD authentication), or a SQL Server authentication contained database user, or a SQL Server authentication user based on a SQL Server authentication login (created in the previous step.) For more information, see

Contained Database Users - Making Your Database Portable.

To create users, connect to the database, and execute statements similar to the following examples:

```
CREATE USER Mary FROM LOGIN Mary;
CREATE USER [mike@contoso.com] FROM EXTERNAL PROVIDER;
```

Initially, only one of the administrators or the owner of the database can create users. To authorize additional users to create new users, grant that selected user the `ALTER ANY USER` permission, by using a statement such as:

```
GRANT ALTER ANY USER TO Mary;
```

To give additional users full control of the database, make them a member of the **db_owner** fixed database role.

In Azure SQL Database use the `ALTER ROLE` statement.

```
ALTER ROLE db_owner ADD MEMBER Mary;
```

In Azure SQL Data Warehouse use [EXEC sp_addrolemember](#).

```
EXEC sp_addrolemember 'db_owner', 'Mary';
```

NOTE

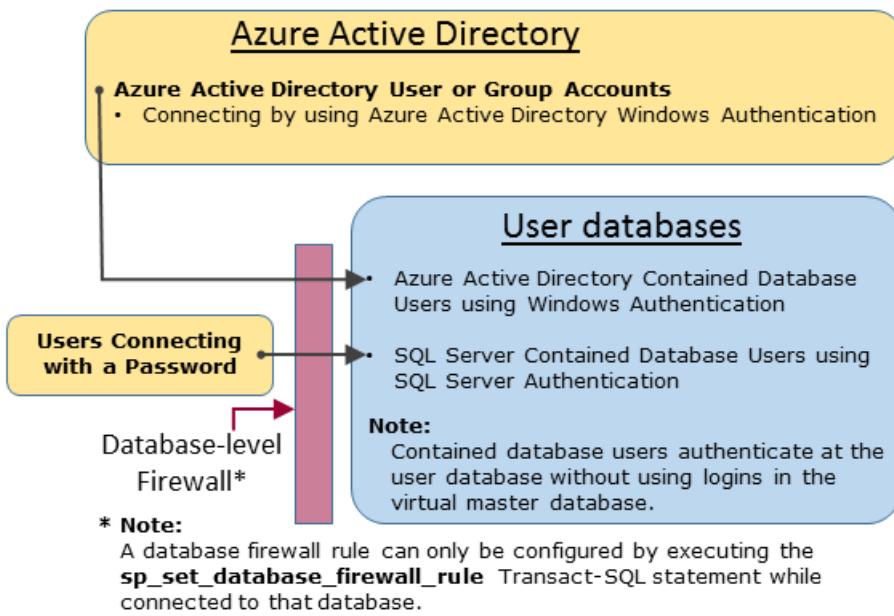
One common reason to create a database user based on a SQL Database server login is for users that need access to multiple databases. Since contained database users are individual entities, each database maintains its own user and its own password. This can cause overhead as the user must then remember each password for each database, and it can become untenable when having to change multiple passwords for many databases. However, when using SQL Server Logins and high availability (active geo-replication and failover groups), the SQL Server logins must be set manually at each server. Otherwise, the database user will no longer be mapped to the server login after a failover occurs, and will not be able to access the database post failover. For more information on configuring logins for geo-replication, please see [Configure and manage Azure SQL Database security for geo-restore or failover](#).

Configuring the database-level firewall

As a best practice, non-administrator users should only have access through the firewall to the databases that they use. Instead of authorizing their IP addresses through the server-level firewall and giving them access to all databases, use the `sp_set_database_firewall_rule` statement to configure the database-level firewall. The database-level firewall cannot be configured by using the portal.

Non-administrator access path

When the database-level firewall is properly configured, the database users can connect using client tools such as SQL Server Management Studio or SQL Server Data Tools. Only the latest tools provide all the features and capabilities. The following diagram shows a typical non-administrator access path.



Groups and roles

Efficient access management uses permissions assigned to groups and roles instead of individual users.

- When using Azure Active Directory authentication, put Azure Active Directory users into an Azure Active Directory group. Create a contained database user for the group. Place one or more database users into a [database role](#) and then assign [permissions](#) to the database role.
- When using SQL Server authentication, create contained database users in the database. Place one or more database users into a [database role](#) and then assign [permissions](#) to the database role.

The database roles can be the built-in roles such as **db_owner**, **db_ddladmin**, **db_datawriter**, **db_datareader**, **db_denydatawriter**, and **db_denydatareader**. **db_owner** is commonly used to grant full permission to only a few users. The other fixed database roles are useful for getting a simple database in development quickly, but are not recommended for most production databases. For example, the **db_datareader** fixed database role grants read access to every table in the database, which is usually more than is strictly necessary. It is far better to use the [CREATE ROLE](#) statement to create your own user-defined database roles and carefully grant each role the least permissions necessary for the business need. When a user is a member of multiple roles, they aggregate the permissions of them all.

Permissions

There are over 100 permissions that can be individually granted or denied in SQL Database. Many of these permissions are nested. For example, the `UPDATE` permission on a schema includes the `UPDATE` permission on each table within that schema. As in most permission systems, the denial of a permission overrides a grant. Because of the nested nature and the number of permissions, it can take careful study to design an appropriate permission system to properly protect your database. Start with the list of permissions at [Permissions \(Database Engine\)](#) and review the [poster size graphic](#) of the permissions.

Considerations and restrictions

When managing logins and users in SQL Database, consider the following:

- You must be connected to the **master** database when executing the `CREATE/ALTER/DROP DATABASE` statements.
- The database user corresponding to the **Server admin** login cannot be altered or dropped.
- US-English is the default language of the **Server admin** login.

- Only the administrators (**Server admin** login or Azure AD administrator) and the members of the **dbmanager** database role in the **master** database have permission to execute the `CREATE DATABASE` and `DROP DATABASE` statements.
- You must be connected to the master database when executing the `CREATE/ALTER/DROP LOGIN` statements. However using logins is discouraged. Use contained database users instead.
- To connect to a user database, you must provide the name of the database in the connection string.
- Only the server-level principal login and the members of the **loginmanager** database role in the **master** database have permission to execute the `CREATE LOGIN`, `ALTER LOGIN`, and `DROP LOGIN` statements.
- When executing the `CREATE/ALTER/DROP LOGIN` and `CREATE/ALTER/DROP DATABASE` statements in an ADO.NET application, using parameterized commands is not allowed. For more information, see [Commands and Parameters](#).
- When executing the `CREATE/ALTER/DROP DATABASE` and `CREATE/ALTER/DROP LOGIN` statements, each of these statements must be the only statement in a Transact-SQL batch. Otherwise, an error occurs. For example, the following Transact-SQL checks whether the database exists. If it exists, a `DROP DATABASE` statement is called to remove the database. Because the `DROP DATABASE` statement is not the only statement in the batch, executing the following Transact-SQL statement results in an error.

```
IF EXISTS (SELECT [name]
           FROM   [sys].[databases]
           WHERE  [name] = N'database_name')
DROP DATABASE [database_name];
GO
```

Instead, use the following Transact-SQL statement:

```
DROP DATABASE IF EXISTS [database_name]
```

- When executing the `CREATE USER` statement with the `FOR/FROM LOGIN` option, it must be the only statement in a Transact-SQL batch.
- When executing the `ALTER USER` statement with the `WITH LOGIN` option, it must be the only statement in a Transact-SQL batch.
- To `CREATE/ALTER/DROP` a user requires the `ALTER ANY USER` permission on the database.
- When the owner of a database role tries to add or remove another database user to or from that database role, the following error may occur: **User or role 'Name' does not exist in this database**. This error occurs because the user is not visible to the owner. To resolve this issue, grant the role owner the `VIEW DEFINITION` permission on the user.

Next steps

- To learn more about firewall rules, see [Azure SQL Database Firewall](#).
- For an overview of all the SQL Database security features, see [SQL security overview](#).
- For a tutorial, see [Secure your Azure SQL Database](#).
- For information about views and stored procedures, see [Creating views and stored procedures](#)
- For information about granting access to a database object, see [Granting Access to a Database Object](#)

Use Azure Active Directory Authentication for authentication with SQL

11/7/2019 • 9 minutes to read • [Edit Online](#)

Azure Active Directory authentication is a mechanism of connecting to Azure [SQL Database](#), [Managed Instance](#), and [SQL Data Warehouse](#) by using identities in Azure Active Directory (Azure AD).

NOTE

This topic applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

With Azure AD authentication, you can centrally manage the identities of database users and other Microsoft services in one central location. Central ID management provides a single place to manage database users and simplifies permission management. Benefits include the following:

- It provides an alternative to SQL Server authentication.
- Helps stop the proliferation of user identities across database servers.
- Allows password rotation in a single place.
- Customers can manage database permissions using external (Azure AD) groups.
- It can eliminate storing passwords by enabling integrated Windows authentication and other forms of authentication supported by Azure Active Directory.
- Azure AD authentication uses contained database users to authenticate identities at the database level.
- Azure AD supports token-based authentication for applications connecting to SQL Database.
- Azure AD authentication supports ADFS (domain federation) or native user/password authentication for a local Azure Active Directory without domain synchronization.
- Azure AD supports connections from SQL Server Management Studio that use Active Directory Universal Authentication, which includes Multi-Factor Authentication (MFA). MFA includes strong authentication with a range of easy verification options — phone call, text message, smart cards with pin, or mobile app notification. For more information, see [SSMS support for Azure AD MFA with SQL Database and SQL Data Warehouse](#).
- Azure AD supports similar connections from SQL Server Data Tools (SSDT) that use Active Directory Interactive Authentication. For more information, see [Azure Active Directory support in SQL Server Data Tools \(SSDT\)](#).

NOTE

Connecting to SQL Server running on an Azure VM is not supported using an Azure Active Directory account. Use a domain Active Directory account instead.

The configuration steps include the following procedures to configure and use Azure Active Directory authentication.

1. Create and populate Azure AD.
2. Optional: Associate or change the active directory that is currently associated with your Azure Subscription.
3. Create an Azure Active Directory administrator for the Azure SQL Database server, the Managed Instance,

- or the [Azure SQL Data Warehouse](#).
4. Configure your client computers.
 5. Create contained database users in your database mapped to Azure AD identities.
 6. Connect to your database by using Azure AD identities.

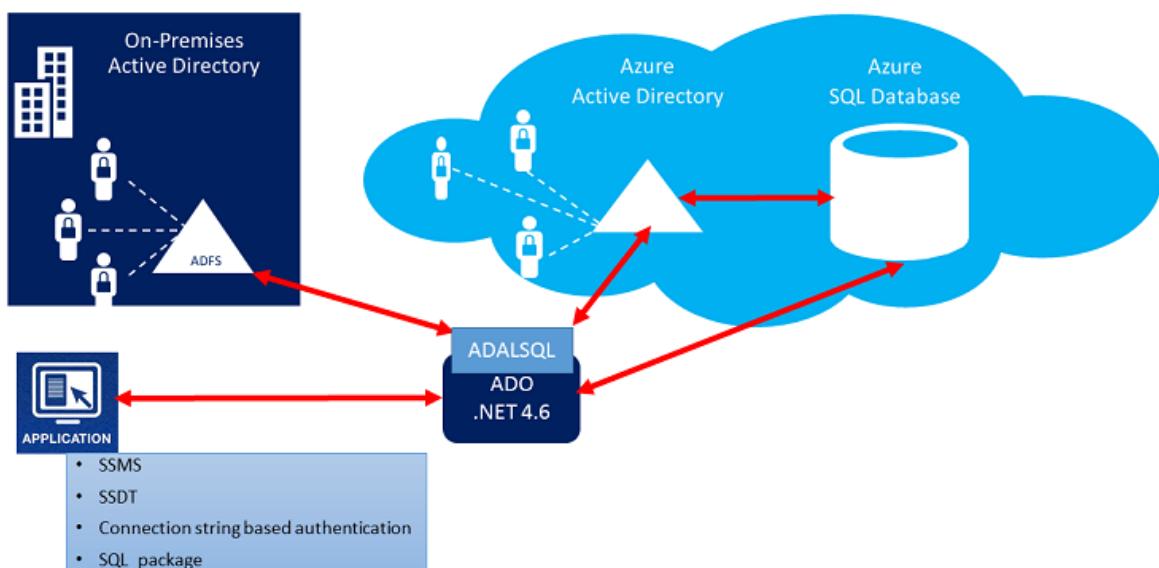
NOTE

To learn how to create and populate Azure AD, and then configure Azure AD with Azure SQL Database, Managed Instance, and SQL Data Warehouse, see [Configure Azure AD with Azure SQL Database](#).

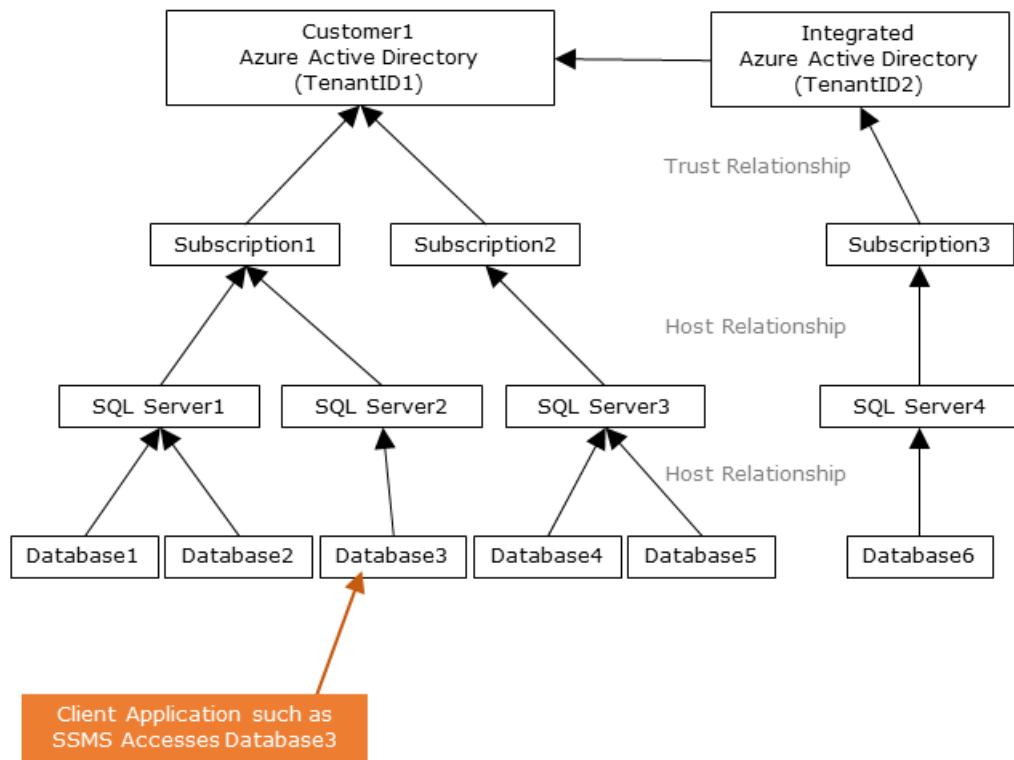
Trust architecture

The following high-level diagram summarizes the solution architecture of using Azure AD authentication with Azure SQL Database. The same concepts apply to SQL Data Warehouse. To support Azure AD native user password, only the Cloud portion and Azure AD/Azure SQL Database is considered. To support Federated authentication (or user/password for Windows credentials), the communication with ADFS block is required. The arrows indicate communication pathways.

Azure AD Authentication with SQL V12 DB

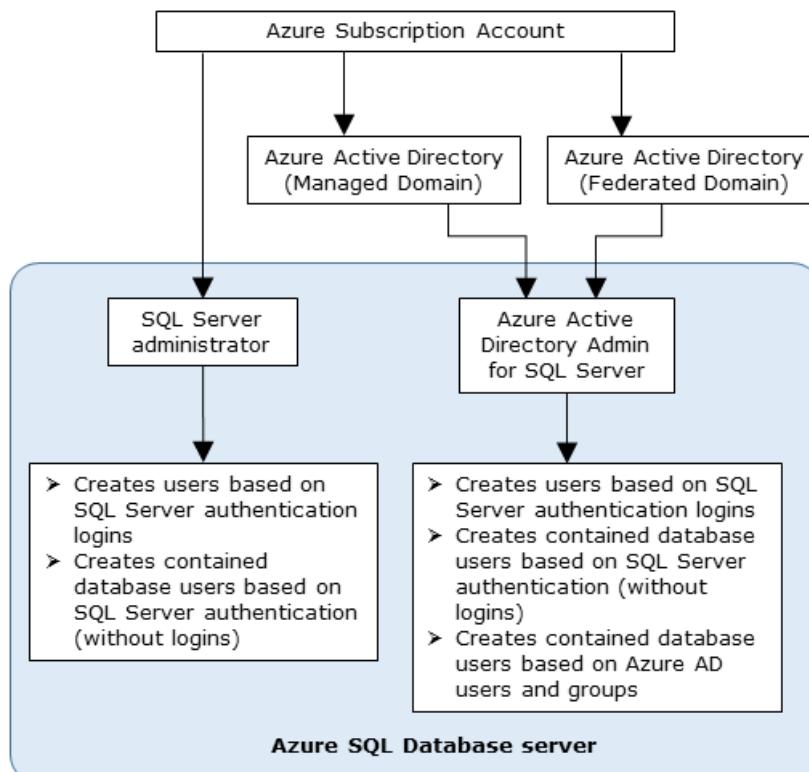


The following diagram indicates the federation, trust, and hosting relationships that allow a client to connect to a database by submitting a token. The token is authenticated by an Azure AD, and is trusted by the database. Customer 1 can represent an Azure Active Directory with native users or an Azure AD with federated users. Customer 2 represents a possible solution including imported users; in this example coming from a federated Azure Active Directory with ADFS being synchronized with Azure Active Directory. It's important to understand that access to a database using Azure AD authentication requires that the hosting subscription is associated to the Azure AD. The same subscription must be used to create the SQL Server hosting the Azure SQL Database or SQL Data Warehouse.



Administrator structure

When using Azure AD authentication, there are two Administrator accounts for the SQL Database server and Managed Instance; the original SQL Server administrator and the Azure AD administrator. The same concepts apply to SQL Data Warehouse. Only the administrator based on an Azure AD account can create the first Azure AD contained database user in a user database. The Azure AD administrator login can be an Azure AD user or an Azure AD group. When the administrator is a group account, it can be used by any group member, enabling multiple Azure AD administrators for the SQL Server instance. Using group account as an administrator enhances manageability by allowing you to centrally add and remove group members in Azure AD without changing the users or permissions in SQL Database. Only one Azure AD administrator (a user or group) can be configured at any time.



Permissions

To create new users, you must have the `ALTER ANY USER` permission in the database. The `ALTER ANY USER` permission can be granted to any database user. The `ALTER ANY USER` permission is also held by the server administrator accounts, and database users with the `CONTROL ON DATABASE` or `ALTER ON DATABASE` permission for that database, and by members of the `db_owner` database role.

To create a contained database user in Azure SQL Database, Managed Instance, or SQL Data Warehouse, you must connect to the database or instance using an Azure AD identity. To create the first contained database user, you must connect to the database by using an Azure AD administrator (who is the owner of the database). This is demonstrated in [Configure and manage Azure Active Directory authentication with SQL Database or SQL Data Warehouse](#). Any Azure AD authentication is only possible if the Azure AD admin was created for Azure SQL Database or SQL Data Warehouse server. If the Azure Active Directory admin was removed from the server, existing Azure Active Directory users created previously inside SQL Server can no longer connect to the database using their Azure Active Directory credentials.

Azure AD features and limitations

- The following members of Azure AD can be provisioned in Azure SQL server or SQL Data Warehouse:
 - Native members: A member created in Azure AD in the managed domain or in a customer domain. For more information, see [Add your own domain name to Azure AD](#).
 - Federated domain members: A member created in Azure AD with a federated domain. For more information, see [Microsoft Azure now supports federation with Windows Server Active Directory](#).
 - Imported members from other Azure AD's who are native or federated domain members.
 - Active Directory groups created as security groups.
- Azure AD users that are part of a group that has `db_owner` server role cannot use the **CREATE DATABASE SCOPED CREDENTIAL** syntax against Azure SQL Database and Azure SQL Data Warehouse. You will see the following error:

```
SQL Error [2760] [S0001]: The specified schema name 'user@mydomain.com' either does not exist or you do not have permission to use it.
```

Grant the `db_owner` role directly to the individual Azure AD user to mitigate the **CREATE DATABASE SCOPED CREDENTIAL** issue.

- These system functions return NULL values when executed under Azure AD principals:

- `SUSER_ID()`
- `SUSER_NAME(<admin ID>)`
- `SUSER_SNAME(<admin SID>)`
- `SUSER_ID(<admin name>)`
- `SUSER_SID(<admin name>)`

Managed Instances

- Azure AD server principals (logins) and users are supported as a preview feature for [Managed Instances](#).
- Setting Azure AD server principals (logins) mapped to an Azure AD group as database owner is not supported in [Managed Instances](#).
 - An extension of this is that when a group is added as part of the `dbcreator` server role, users from this group can connect to the Managed Instance and create new databases, but will not be able to access the database. This is because the new database owner is SA, and not the Azure AD user. This issue does not manifest if the individual user is added to the `dbcreator` server role.
- SQL Agent management and jobs execution is supported for Azure AD server principals (logins).
- Database backup and restore operations can be executed by Azure AD server principals (logins).

- Auditing of all statements related to Azure AD server principals (logins) and authentication events is supported.
- Dedicated administrator connection for Azure AD server principals (logins) which are members of sysadmin server role is supported.
 - Supported through SQLCMD Utility and SQL Server Management Studio.
- Logon triggers are supported for logon events coming from Azure AD server principals (logins).
- Service Broker and DB mail can be setup using an Azure AD server principal (login).

Connecting using Azure AD identities

Azure Active Directory authentication supports the following methods of connecting to a database using Azure AD identities:

- Azure Active Directory Password
- Azure Active Directory Integrated
- Azure Active Directory Universal with MFA
- Using Application token authentication

The following authentication methods are supported for Azure AD server principals (logins) (**public preview**):

- Azure Active Directory Password
- Azure Active Directory Integrated
- Azure Active Directory Universal with MFA

Additional considerations

- To enhance manageability, we recommend you provision a dedicated Azure AD group as an administrator.
- Only one Azure AD administrator (a user or group) can be configured for an Azure SQL Database server or Azure SQL Data Warehouse at any time.
 - The addition of Azure AD server principals (logins) for Managed Instances (**public preview**) allows the possibility of creating multiple Azure AD server principals (logins) that can be added to the `sysadmin` role.
- Only an Azure AD administrator for SQL Server can initially connect to the Azure SQL Database server, Managed Instance, or Azure SQL Data Warehouse using an Azure Active Directory account. The Active Directory administrator can configure subsequent Azure AD database users.
- We recommend setting the connection timeout to 30 seconds.
- SQL Server 2016 Management Studio and SQL Server Data Tools for Visual Studio 2015 (version 14.0.60311.1 April 2016 or later) support Azure Active Directory authentication. (Azure AD authentication is supported by the **.NET Framework Data Provider for SqlServer**; at least version .NET Framework 4.6). Therefore the newest versions of these tools and data-tier applications (DAC and .BACPAC) can use Azure AD authentication.
- Beginning with version 15.0.1, [sqlcmd utility](#) and [bcp utility](#) support Active Directory Interactive authentication with MFA.
- SQL Server Data Tools for Visual Studio 2015 requires at least the April 2016 version of the Data Tools (version 14.0.60311.1). Currently Azure AD users are not shown in SSDT Object Explorer. As a workaround, view the users in `sys.database_principals`.
- [Microsoft JDBC Driver 6.0 for SQL Server](#) supports Azure AD authentication. Also, see [Setting the Connection Properties](#).
- PolyBase cannot authenticate by using Azure AD authentication.
- Azure AD authentication is supported for SQL Database by the Azure portal **Import Database** and **Export Database** blades. Import and export using Azure AD authentication is also supported from the PowerShell command.

- Azure AD authentication is supported for SQL Database, Managed Instance, and SQL Data Warehouse by use CLI. For more information, see [Configure and manage Azure Active Directory authentication with SQL Database or SQL Data Warehouse](#) and [SQL Server - az sql server](#).

Next steps

- To learn how to create and populate Azure AD, and then configure Azure AD with Azure SQL Database or Azure SQL Data Warehouse, see [Configure and manage Azure Active Directory authentication with SQL Database, Managed Instance, or SQL Data Warehouse](#).
- For a tutorial of using Azure AD server principals (logins) with Managed Instances, see [Azure AD server principals \(logins\) with Managed Instances](#)
- For an overview of access and control in SQL Database, see [SQL Database access and control](#).
- For an overview of logins, users, and database roles in SQL Database, see [Logins, users, and database roles](#).
- For more information about database principals, see [Principals](#).
- For more information about database roles, see [Database roles](#).
- For syntax on creating Azure AD server principals (logins) for Managed Instances, see [CREATE LOGIN](#).
- For more information about firewall rules in SQL Database, see [SQL Database firewall rules](#).

Configure and manage Azure Active Directory authentication with SQL

1/8/2020 • 26 minutes to read • [Edit Online](#)

This article shows you how to create and populate Azure AD, and then use Azure AD with Azure [SQL Database](#), [managed instance](#), and [SQL Data Warehouse](#). For an overview, see [Azure Active Directory Authentication](#).

NOTE

This article applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

IMPORTANT

Connecting to SQL Server running on an Azure VM is not supported using an Azure Active Directory account. Use a domain Active Directory account instead.

Create and populate an Azure AD

Create an Azure AD and populate it with users and groups. Azure AD can be the initial Azure AD managed domain. Azure AD can also be an on-premises Active Directory Domain Services that is federated with the Azure AD.

For more information, see [Integrating your on-premises identities with Azure Active Directory](#), [Add your own domain name to Azure AD](#), [Microsoft Azure now supports federation with Windows Server Active Directory](#), [Administering your Azure AD directory](#), [Manage Azure AD using Windows PowerShell](#), and [Hybrid Identity Required Ports and Protocols](#).

Associate or add an Azure subscription to Azure Active Directory

1. Associate your Azure subscription to Azure Active Directory by making the directory a trusted directory for the Azure subscription hosting the database. For details, see [How Azure subscriptions are associated with Azure AD](#).
2. Use the directory switcher in the Azure portal to switch to the subscription associated with domain.

IMPORTANT

Every Azure subscription has a trust relationship with an Azure AD instance. This means that it trusts that directory to authenticate users, services, and devices. Multiple subscriptions can trust the same directory, but a subscription trusts only one directory. This trust relationship that a subscription has with a directory is unlike the relationship that a subscription has with all other resources in Azure (websites, databases, and so on), which are more like child resources of a subscription. If a subscription expires, then access to those other resources associated with the subscription also stops. But the directory remains in Azure, and you can associate another subscription with that directory and continue to manage the directory users. For more information about resources, see [Understanding resource access in Azure](#). To learn more about this trusted relationship see [How to associate or add an Azure subscription to Azure Active Directory](#).

Create an Azure AD administrator for Azure SQL server

Each Azure SQL server (which hosts a SQL Database or SQL Data Warehouse) starts with a single server administrator account that is the administrator of the entire Azure SQL server. A second SQL Server administrator must be created, that is an Azure AD account. This principal is created as a contained database user in the master database. As administrators, the server administrator accounts are members of the **db_owner** role in every user database, and enter each user database as the **dbo** user. For more information about the server administrator accounts, see [Managing Databases and Logins in Azure SQL Database](#).

When using Azure Active Directory with geo-replication, the Azure Active Directory administrator must be configured for both the primary and the secondary servers. If a server does not have an Azure Active Directory administrator, then Azure Active Directory logins and users receive a "Cannot connect" to server error.

NOTE

Users that are not based on an Azure AD account (including the Azure SQL server administrator account), cannot create Azure AD-based users, because they do not have permission to validate proposed database users with the Azure AD.

Provision an Azure Active Directory administrator for your managed instance

IMPORTANT

Only follow these steps if you are provisioning a managed instance. This operation can only be executed by Global/Company administrator or a Privileged Role Administrator in Azure AD. Following steps describe the process of granting permissions for users with different privileges in directory.

NOTE

For Azure AD admins for MI created prior to GA, but continue operating post GA, there is no functional change to the existing behavior. For more information, see the [New Azure AD admin functionality for MI](#) section for more details.

Your managed instance needs permissions to read Azure AD to successfully accomplish tasks such as authentication of users through security group membership or creation of new users. For this to work, you need to grant permissions to managed instance to read Azure AD. There are two ways to do it: from Portal and PowerShell. The following steps both methods.

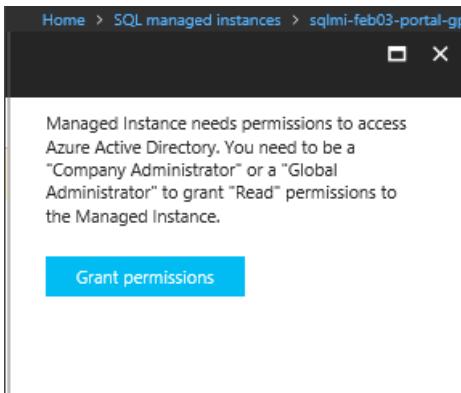
1. In the Azure portal, in the upper-right corner, select your connection to drop down a list of possible Active Directories.
2. Choose the correct Active Directory as the default Azure AD.

This step links the subscription associated with Active Directory with Managed Instance making sure that the same subscription is used for both Azure AD and the Managed Instance.

3. Navigate to Managed Instance and select one that you want to use for Azure AD integration.

The screenshot shows the Azure portal interface for managing SQL managed instances. On the left, a sidebar lists various managed instances, including 'mi-workshop2' which is highlighted with a red box. The main pane displays the 'mi-workshop2 - Active Directory admin' settings. At the top, there are buttons for 'Set admin', 'Remove admin', and 'Save'. A yellow banner at the top right states: 'Managed Instance needs permissions to access Azure Active Directory. Click here to grant "Read" permissions to your Managed Instance.' Below the banner, it says 'Azure Active Directory authentication allows you to centrally manage identity and access to your Managed instance.' and provides a 'Learn more' link. The 'Active Directory admin' section shows a status of 'No Active Directory admin'. Other sections include 'Overview', 'Activity log', 'Tags', 'Diagnose and solve problems', 'SETTINGS' (with 'Quick start' and 'Connection strings' options), 'Pricing tier', 'Locks', 'Automation script', 'SUPPORT + TROUBLESHOOTING' (with 'Resource health' and 'New support request' options), and a 'Grant permissions' button.

4. Select the banner on top of the Active Directory admin page and grant permission to the current user. If you're logged in as Global/Company administrator in Azure AD, you can do it from the Azure portal or using PowerShell with the script below.



```

# Gives Azure Active Directory read permission to a Service Principal representing the managed
instance.

# Can be executed only by a "Company Administrator", "Global Administrator", or "Privileged Role
Administrator" type of user.

$aadTenant = "<YourTenantId>" # Enter your tenant ID
$managedInstanceName = "MyManagedInstance"

# Get Azure AD role "Directory Users" and create if it doesn't exist
$roleName = "Directory Readers"
$role = Get-AzureADDirectoryRole | Where-Object {$__.displayName -eq $roleName}
if ($role -eq $null) {
    # Instantiate an instance of the role template
    $roleTemplate = Get-AzureADDirectoryRoleTemplate | Where-Object {$__.displayName -eq $roleName}
    Enable-AzureADDirectoryRole -RoleTemplateId $roleTemplate.ObjectId
    $role = Get-AzureADDirectoryRole | Where-Object {$__.displayName -eq $roleName}
}

# Get service principal for managed instance
$roleMember = Get-AzureADServicePrincipal -SearchString $managedInstanceName
$roleMember.Count
if ($roleMember -eq $null) {
    Write-Output "Error: No Service Principals with name '$($managedInstanceName)', make sure that
managedInstanceName parameter was      entered correctly."
    exit
}
if (-not ($roleMember.Count -eq 1)) {
    Write-Output "Error: More than one service principal with name pattern '$
($managedInstanceName)'"
    Write-Output "Dumping selected service principals...."
    $roleMember
    exit
}

# Check if service principal is already member of readers role
$allDirReaders = Get-AzureADDirectoryRoleMember -ObjectId $role.ObjectId
$selDirReader = $allDirReaders | where{$__.ObjectId -match      $roleMember.ObjectId}

if ($selDirReader -eq $null) {
    # Add principal to readers role
    Write-Output "Adding service principal '$($managedInstanceName)' to      'Directory Readers'
role'..."
    Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId      $roleMember.ObjectId
    Write-Output "'$($managedInstanceName)' service principal added to      'Directory Readers'
role'..."

    #Write-Output "Dumping service principal '$($managedInstanceName)':"
    #$allDirReaders = Get-AzureADDirectoryRoleMember -ObjectId $role.ObjectId
    #$allDirReaders | where{$__.ObjectId -match $roleMember.ObjectId}
}
else {
    Write-Output "Service principal '$($managedInstanceName)' is already      member of 'Directory
Readers' role."
}

```

5. After the operation is successfully completed, the following notification will show up in the top-right corner:



6. Now you can choose your Azure AD admin for your managed instance. For that, on the Active Directory admin page, select **Set admin** command.

The screenshot shows the Azure portal interface for managing a SQL managed instance named 'sqlmi-feb03-portal-gp8'. In the top navigation bar, the path 'Home > SQL managed instances > sqlmi-feb03-portal-gp8 - Active Directory admin' is visible. The main content area is titled 'sqlmi-feb03-portal-gp8 - Active Directory admin' and contains a sub-header 'Azure Active Directory authentication allows you to centrally manage identity and access to your Managed instance.' Below this, there is a section titled 'Active Directory admin' with a note 'No Active Directory admin'. On the left side, there is a sidebar with various options like Overview, Activity log, Tags, Diagnose and solve problems, Connection strings, Pricing tier, Locks, and Automation script. The 'Connection strings' option is highlighted with a red box. At the bottom of the sidebar, there is a link 'Learn more'.

7. In the AAD admin page, search for a user, select the user or group to be an administrator, and then select **Select**.

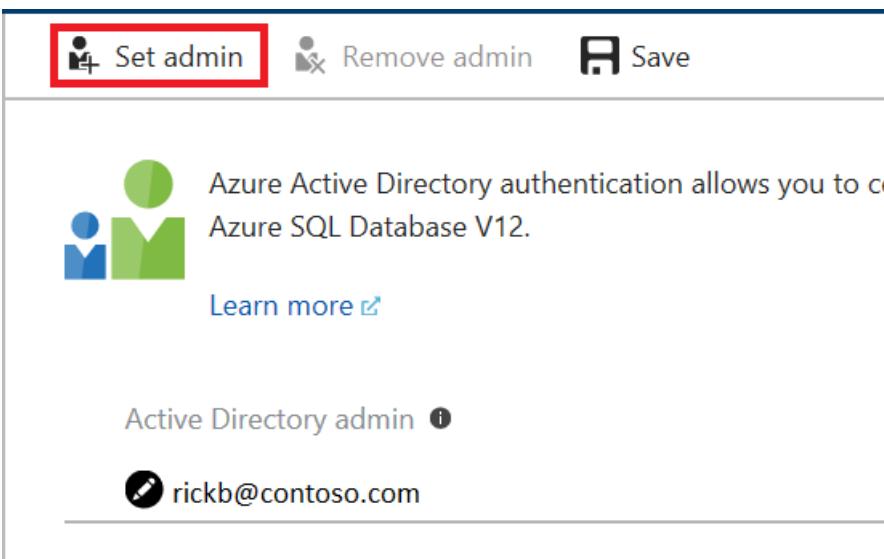
The Active Directory admin page shows all members and groups of your Active Directory. Users or groups that are grayed out can't be selected because they aren't supported as Azure AD administrators. See the list of supported admins in [Azure AD Features and Limitations](#). Role-based access control (RBAC) applies only to the Azure portal and isn't propagated to SQL Server.

The screenshot shows the 'Add admin' dialog box within the Azure portal. The title bar says 'Microsoft Azure' and the URL is '... > SQL servers > test_srv - Active Directory admin > Add admin'. The main area is titled 'Add admin' and 'Active Directory admin'. It includes a 'Select' button and a search input field containing 'rick b'. Below the search input, a list of users is shown:

- Rick B (selected, highlighted with a red box), rickb@contoso.com
- Rick F, rickf@contoso.com
- Rick M, rickm@contoso.com

At the bottom, there is a 'Selected' section containing 'Rick B' and a large blue 'Select' button.

8. At the top of the Active Directory admin page, select **Save**.



The screenshot shows a user interface for managing database administrators. At the top, there are three buttons: 'Set admin' (highlighted with a red box), 'Remove admin', and 'Save'. Below these buttons, there is a section titled 'Azure Active Directory authentication allows you to connect to your database using your Azure Active Directory users and groups.' It includes a 'Learn more' link. Further down, it shows the 'Active Directory admin' field containing 'rickb@contoso.com' with a pencil icon indicating it's editable.

The process of changing the administrator may take several minutes. Then the new administrator appears in the Active Directory admin box.

After provisioning an Azure AD admin for your managed instance, you can begin to create Azure AD server principals (logins) with the [CREATE LOGIN](#) syntax. For more information, see [managed instance overview](#).

TIP

To later remove an Admin, at the top of the Active Directory admin page, select **Remove admin**, and then select **Save**.

New Azure AD admin functionality for MI

The table below summarizes the functionality for the public preview Azure AD login admin for MI, versus a new functionality delivered with GA for Azure AD logins.

AZURE AD LOGIN ADMIN FOR MI DURING PUBLIC PREVIEW	GA FUNCTIONALITY FOR AZURE AD ADMIN FOR MI
Behaves in a similar way as Azure AD admin for SQL Database, which enables Azure AD authentication, but the Azure AD admin cannot create Azure AD or SQL logins in the master db for MI.	Azure AD admin has sysadmin permission and can create AAD and SQL logins in master db for MI.
Is not present in the sys.server_principals view	Is present in the sys.server_principals view
Allows individual Azure AD guest users to be set up as Azure AD admin for MI. For more information, see Add Azure Active Directory B2B collaboration users in the Azure portal .	Requires creation of an Azure AD group with guest users as members to set up this group as an Azure AD admin for MI. For more information, see Azure AD business to business support .

As a best practice for existing Azure AD admins for MI created before GA, and still operating post GA, reset the Azure AD admin using the Azure portal "Remove admin" and "Set admin" option for the same Azure AD user or group.

Known issues with the Azure AD login GA for MI

- If an Azure AD login exists in the master database for MI, created using the T-SQL command `CREATE LOGIN [myaadaccount] FROM EXTERNAL PROVIDER`, it can't be set up as an Azure AD admin for MI. You'll experience an error setting the login as an Azure AD admin using the Azure portal, PowerShell, or CLI commands to create the Azure AD login.
 - The login must be dropped in the master database using the command `DROP LOGIN [myaadaccount]`, before the account can be created as an Azure AD admin.

- Set up the Azure AD admin account in the Azure portal after the `DROP LOGIN` succeeds.
- If you can't set up the Azure AD admin account, check in the master database of the managed instance for the login. Use the following command: `SELECT * FROM sys.server_principals`
- Setting up an Azure AD admin for MI will automatically create a login in the master database for this account. Removing the Azure AD admin will automatically drop the login from the master database.
- Individual Azure AD guest users are not supported as Azure AD admins for MI. Guest users must be part of an Azure AD group to be set up as Azure AD admin. Currently, the Azure portal blade doesn't gray out guest users for another Azure AD, allowing users to continue with the admin setup. Saving guest users as an Azure AD admin will cause the setup to fail.
 - If you wish to make a guest user an Azure AD admin for MI, include the guest user in an Azure AD group, and set this group as an Azure AD admin.

PowerShell for SQL managed instance

- [PowerShell](#)
- [Azure CLI](#)

To run PowerShell cmdlets, you need to have Azure PowerShell installed and running. For detailed information, see [How to install and configure Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

To provision an Azure AD admin, execute the following Azure PowerShell commands:

- `Connect-AzAccount`
- `Select-AzSubscription`

Cmdlets used to provision and manage Azure AD admin for SQL managed instance:

CMDLET NAME	DESCRIPTION
Set-AzSqlInstanceActiveDirectoryAdministrator	Provisions an Azure AD administrator for SQL managed instance in the current subscription. (Must be from the current subscription)
Remove-AzSqlInstanceActiveDirectoryAdministrator	Removes an Azure AD administrator for SQL managed instance in the current subscription.
Get-AzSqlInstanceActiveDirectoryAdministrator	Returns information about an Azure AD administrator for SQL managed instance in the current subscription.

The following command gets information about an Azure AD administrator for a managed instance named ManagedInstance01 that is associated with a resource group named ResourceGroup01.

```
Get-AzSqlInstanceActiveDirectoryAdministrator -ResourceGroupName "ResourceGroup01" -InstanceName "ManagedInstance01"
```

The following command provisions an Azure AD administrator group named DBAs for the managed instance named ManagedInstance01. This server is associated with resource group ResourceGroup01.

```
Set-AzSqlInstanceActiveDirectoryAdministrator -ResourceGroupName "ResourceGroup01" -InstanceName "ManagedInstance01" -DisplayName "DBAs" -ObjectId "40b79501-b343-44ed-9ce7-da4c8cc7353b"
```

The following command removes the Azure AD administrator for the managed instance named ManagedInstanceName01 associated with the resource group ResourceGroup01.

```
Remove-AzSqlInstanceActiveDirectoryAdministrator -ResourceGroupName "ResourceGroup01" -InstanceName "ManagedInstanceName01" -Confirm -PassThru
```

Provision an Azure Active Directory administrator for your Azure SQL Database server

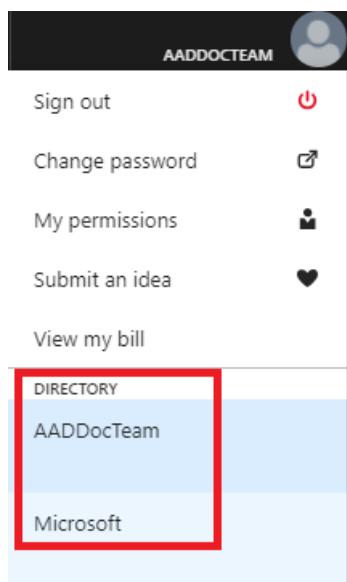
IMPORTANT

Only follow these steps if you are provisioning an Azure SQL Database server or Data Warehouse.

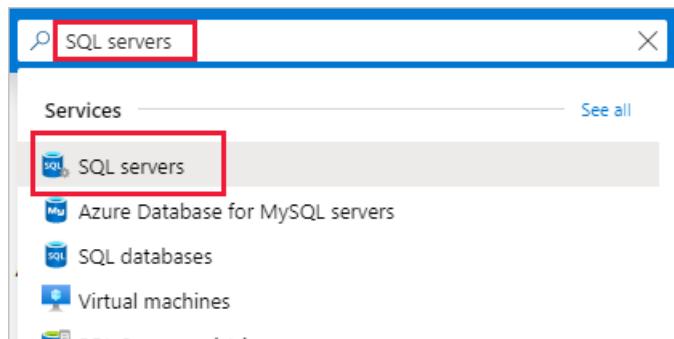
The following two procedures show you how to provision an Azure Active Directory administrator for your Azure SQL server in the Azure portal and by using PowerShell.

Azure portal

1. In the [Azure portal](#), in the upper-right corner, select your connection to drop down a list of possible Active Directories. Choose the correct Active Directory as the default Azure AD. This step links the subscription-associated Active Directory with Azure SQL server making sure that the same subscription is used for both Azure AD and SQL Server. (The Azure SQL server can be hosting either Azure SQL Database or Azure SQL Data Warehouse.)



2. Search for and select **SQL server**.



NOTE

On this page, before you select **SQL servers**, you can select the **star** next to the name to *favorite* the category and add **SQL servers** to the left navigation bar.

3. On **SQL Server** page, select **Active Directory admin**.

4. In the **Active Directory admin** page, select **Set admin**.

The screenshot shows the 'test_srv - Active Directory admin' page. At the top, there's a 'Search resources, services, and docs (G+ /)' bar. Below it, the navigation path is 'Home > SQL servers > test_srv - Active Directory admin'. The main area has a 'SQL servers' section on the left with a 'Contoso' dropdown, an 'Add' button, and a 'test_srv' item selected and highlighted with a red box. To the right, there's a 'test_srv - Active Directory admin' section with a 'Search (Ctrl+ /)' input field. A 'Set admin' button is highlighted with a red box. Below it are links for 'Diagnose and solve problems', 'Settings', 'Quick start', 'Failover groups', 'Manage Backups', and 'Active Directory admin' (which is also highlighted with a red box). Further down are links for 'SQL databases', 'SQL elastic pools', 'Deleted databases', and 'Import/Export history'. A note about Azure Active Directory authentication is present, along with 'Learn more' and 'Active Directory admin' status indicators.

5. In the **Add admin** page, search for a user, select the user or group to be an administrator, and then select **Select**. (The Active Directory admin page shows all members and groups of your Active Directory. Users or groups that are grayed out cannot be selected because they are not supported as Azure AD administrators. (See the list of supported admins in the **Azure AD Features and Limitations** section of [Use Azure Active Directory Authentication for authentication with SQL Database or SQL Data Warehouse](#).) Role-based access control (RBAC) applies only to the portal and is not propagated to SQL Server.

The screenshot shows the 'Add admin' dialog in the Microsoft Azure portal. The search bar at the top contains 'rick b'. Below it, a list of users is displayed. The user 'Rick B rickb@contoso.com' is selected and highlighted with a red box. Other users listed are 'Rick F rickf@contoso.com' and 'Rick M rickm@contoso.com'. Below the list, a 'Selected' section contains 'Rick B'. At the bottom of the dialog is a blue 'Select' button, which is also highlighted with a red box.

6. At the top of the **Active Directory admin** page, select **SAVE**.

The screenshot shows the 'Active Directory admin' page in the Microsoft Azure portal. At the top, there are three buttons: 'Set admin' (highlighted with a red box), 'Remove admin', and 'Save'. Below the buttons, there is a note about Azure Active Directory authentication. Underneath the note, the 'Active Directory admin' section shows 'rickb@contoso.com'.

The process of changing the administrator may take several minutes. Then the new administrator appears in the **Active Directory admin** box.

NOTE

When setting up the Azure AD admin, the new admin name (user or group) cannot already be present in the virtual master database as a SQL Server authentication user. If present, the Azure AD admin setup will fail; rolling back its creation and indicating that such an admin (name) already exists. Since such a SQL Server authentication user is not part of the Azure AD, any effort to connect to the server using Azure AD authentication fails.

To later remove an Admin, at the top of the **Active Directory admin** page, select **Remove admin**, and then select **Save**.

PowerShell for Azure SQL Database and Azure SQL Data Warehouse

- [PowerShell](#)

- Azure CLI

To run PowerShell cmdlets, you need to have Azure PowerShell installed and running. For detailed information, see [How to install and configure Azure PowerShell](#). To provision an Azure AD admin, execute the following Azure PowerShell commands:

- Connect-AzAccount
- Select-AzSubscription

Cmdlets used to provision and manage Azure AD admin for Azure SQL Database and Azure SQL Data Warehouse:

CMDLET NAME	DESCRIPTION
Set-AzSqlServerActiveDirectoryAdministrator	Provisions an Azure Active Directory administrator for Azure SQL server or Azure SQL Data Warehouse. (Must be from the current subscription)
Remove-AzSqlServerActiveDirectoryAdministrator	Removes an Azure Active Directory administrator for Azure SQL server or Azure SQL Data Warehouse.
Get-AzSqlServerActiveDirectoryAdministrator	Returns information about an Azure Active Directory administrator currently configured for the Azure SQL server or Azure SQL Data Warehouse.

Use PowerShell command `get-help` to see more information for each of these commands. For example,

```
get-help Set-AzSqlServerActiveDirectoryAdministrator .
```

The following script provisions an Azure AD administrator group named **DBA_Group** (object ID `40b79501-b343-44ed-9ce7-da4c8cc7353f`) for the **demo_server** server in a resource group named **Group-23**:

```
Set-AzSqlServerActiveDirectoryAdministrator -ResourceGroupName "Group-23" -ServerName "demo_server" -  
DisplayName "DBA_Group"
```

The **DisplayName** input parameter accepts either the Azure AD display name or the User Principal Name. For example, `DisplayName="John Smith"` and `DisplayName="johns@contoso.com"`. For Azure AD groups only the Azure AD display name is supported.

NOTE

The Azure PowerShell command `Set-AzSqlServerActiveDirectoryAdministrator` does not prevent you from provisioning Azure AD admins for unsupported users. An unsupported user can be provisioned, but can not connect to a database.

The following example uses the optional **ObjectId**:

```
Set-AzSqlServerActiveDirectoryAdministrator -ResourceGroupName "Group-23" -ServerName "demo_server" `  
-DisplayName "DBA_Group" -ObjectId "40b79501-b343-44ed-9ce7-da4c8cc7353f"
```

NOTE

The Azure AD **ObjectId** is required when the **DisplayName** is not unique. To retrieve the **ObjectId** and **DisplayName** values, use the Active Directory section of Azure Classic Portal, and view the properties of a user or group.

The following example returns information about the current Azure AD admin for Azure SQL server:

```
Get-AzSqlServerActiveDirectoryAdministrator -ResourceGroupName "Group-23" -ServerName "demo_server" | Format-List
```

The following example removes an Azure AD administrator:

```
Remove-AzSqlServerActiveDirectoryAdministrator -ResourceGroupName "Group-23" -ServerName "demo_server"
```

NOTE

You can also provision an Azure Active Directory Administrator by using the REST APIs. For more information, see [Service Management REST API Reference and Operations for Azure SQL Database Operations for Azure SQL Database](#)

Configure your client computers

On all client machines, from which your applications or users connect to Azure SQL Database or Azure SQL Data Warehouse using Azure AD identities, you must install the following software:

- .NET Framework 4.6 or later from <https://msdn.microsoft.com/library/5a4x27ek.aspx>.
- Azure Active Directory Authentication Library for SQL Server (*ADAL.DLL*). Below are the download links to install the latest SSMS, ODBC, and OLE DB driver that contains the *ADAL.DLL* library.
 1. [SQL Server Management Studio](#)
 2. [ODBC Driver 17 for SQL Server](#)
 3. [OLE DB Driver 18 for SQL Server](#)

You can meet these requirements by:

- Installing the latest version of [SQL Server Management Studio](#) or [SQL Server Data Tools](#) meets the .NET Framework 4.6 requirement.
 - SSMS installs the x86 version of *ADAL.DLL*.
 - SSDT installs the amd64 version of *ADAL.DLL*.
 - The latest Visual Studio from [Visual Studio Downloads](#) meets the .NET Framework 4.6 requirement, but does not install the required amd64 version of *ADAL.DLL*.

Create contained database users in your database mapped to Azure AD identities

IMPORTANT

Managed instance now supports Azure AD server principals (logins), which enables you to create logins from Azure AD users, groups, or applications. Azure AD server principals (logins) provides the ability to authenticate to your managed instance without requiring database users to be created as a contained database user. For more information, see [managed instance Overview](#). For syntax on creating Azure AD server principals (logins), see [CREATE LOGIN](#).

Azure Active Directory authentication requires database users to be created as contained database users. A contained database user based on an Azure AD identity, is a database user that does not have a login in the master database, and which maps to an identity in the Azure AD directory that is associated with the database. The Azure AD identity can be either an individual user account or a group. For more information about contained database users, see [Contained Database Users- Making Your Database Portable](#).

NOTE

Database users (with the exception of administrators) cannot be created using the Azure portal. RBAC roles are not propagated to SQL Server, SQL Database, or SQL Data Warehouse. Azure RBAC roles are used for managing Azure Resources, and do not apply to database permissions. For example, the **SQL Server Contributor** role does not grant access to connect to the SQL Database or SQL Data Warehouse. The access permission must be granted directly in the database using Transact-SQL statements.

WARNING

Special characters like colon `:` or ampersand `&` when included as user names in the T-SQL CREATE LOGIN and CREATE USER statements are not supported.

To create an Azure AD-based contained database user (other than the server administrator that owns the database), connect to the database with an Azure AD identity, as a user with at least the **ALTER ANY USER** permission. Then use the following Transact-SQL syntax:

```
CREATE USER <Azure_AD_principal_name> FROM EXTERNAL PROVIDER;
```

Azure_AD_principal_name can be the user principal name of an Azure AD user or the display name for an Azure AD group.

Examples: To create a contained database user representing an Azure AD federated or managed domain user:

```
CREATE USER [bob@contoso.com] FROM EXTERNAL PROVIDER;
CREATE USER [alice@fabrikam.onmicrosoft.com] FROM EXTERNAL PROVIDER;
```

To create a contained database user representing an Azure AD or federated domain group, provide the display name of a security group:

```
CREATE USER [ICU Nurses] FROM EXTERNAL PROVIDER;
```

To create a contained database user representing an application that connects using an Azure AD token:

```
CREATE USER [appName] FROM EXTERNAL PROVIDER;
```

NOTE

This command requires that SQL access Azure AD (the "external provider") on behalf of the logged-in user. Sometimes, circumstances will arise that cause Azure AD to return an exception back to SQL. In these cases, the user will see SQL error 33134, which should contain the AAD-specific error message. Most of the time, the error will say that access is denied, or that the user must enroll in MFA to access the resource, or that access between first-party applications must be handled via preauthorization. In the first two cases, the issue is usually caused by Conditional Access policies that are set in the user's AAD tenant: they prevent the user from accessing the external provider. Updating the CA policies to allow access to the application '00000002-0000-0000-c000-000000000000' (the application ID of the AAD Graph API) should resolve the issue. In the case that the error says access between first-party applications must be handled via preauthorization, the issue is because the user is signed in as a service principal. The command should succeed if it is executed by a user instead.

TIP

You cannot directly create a user from an Azure Active Directory other than the Azure Active Directory that is associated with your Azure subscription. However, members of other Active Directories that are imported users in the associated Active Directory (known as external users) can be added to an Active Directory group in the tenant Active Directory. By creating a contained database user for that AD group, the users from the external Active Directory can gain access to SQL Database.

For more information about creating contained database users based on Azure Active Directory identities, see [CREATE USER \(Transact-SQL\)](#).

NOTE

Removing the Azure Active Directory administrator for Azure SQL server prevents any Azure AD authentication user from connecting to the server. If necessary, unusable Azure AD users can be dropped manually by a SQL Database administrator.

NOTE

If you receive a **Connection Timeout Expired**, you may need to set the `TransparentNetworkIPResolution` parameter of the connection string to false. For more information, see [Connection timeout issue with .NET Framework 4.6.1 - TransparentNetworkIPResolution](#).

When you create a database user, that user receives the **CONNECT** permission and can connect to that database as a member of the **PUBLIC** role. Initially the only permissions available to the user are any permissions granted to the **PUBLIC** role, or any permissions granted to any Azure AD groups that they are a member of. Once you provision an Azure AD-based contained database user, you can grant the user additional permissions, the same way as you grant permission to any other type of user. Typically grant permissions to database roles, and add users to roles. For more information, see [Database Engine Permission Basics](#). For more information about special SQL Database roles, see [Managing Databases and Logins in Azure SQL Database](#). A federated domain user account that is imported into a managed domain as an external user, must use the managed domain identity.

NOTE

Azure AD users are marked in the database metadata with type E (EXTERNAL_USER) and for groups with type X (EXTERNAL_GROUPS). For more information, see [sys.database_principals](#).

Connect to the user database or data warehouse by using SSMS or SSDT

To confirm the Azure AD administrator is properly set up, connect to the **master** database using the Azure AD administrator account. To provision an Azure AD-based contained database user (other than the server administrator that owns the database), connect to the database with an Azure AD identity that has access to the database.

IMPORTANT

Support for Azure Active Directory authentication is available with [SQL Server 2016 Management Studio](#) and [SQL Server Data Tools](#) in Visual Studio 2015. The August 2016 release of SSMS also includes support for Active Directory Universal Authentication, which allows administrators to require Multi-Factor Authentication using a phone call, text message, smart cards with pin, or mobile app notification.

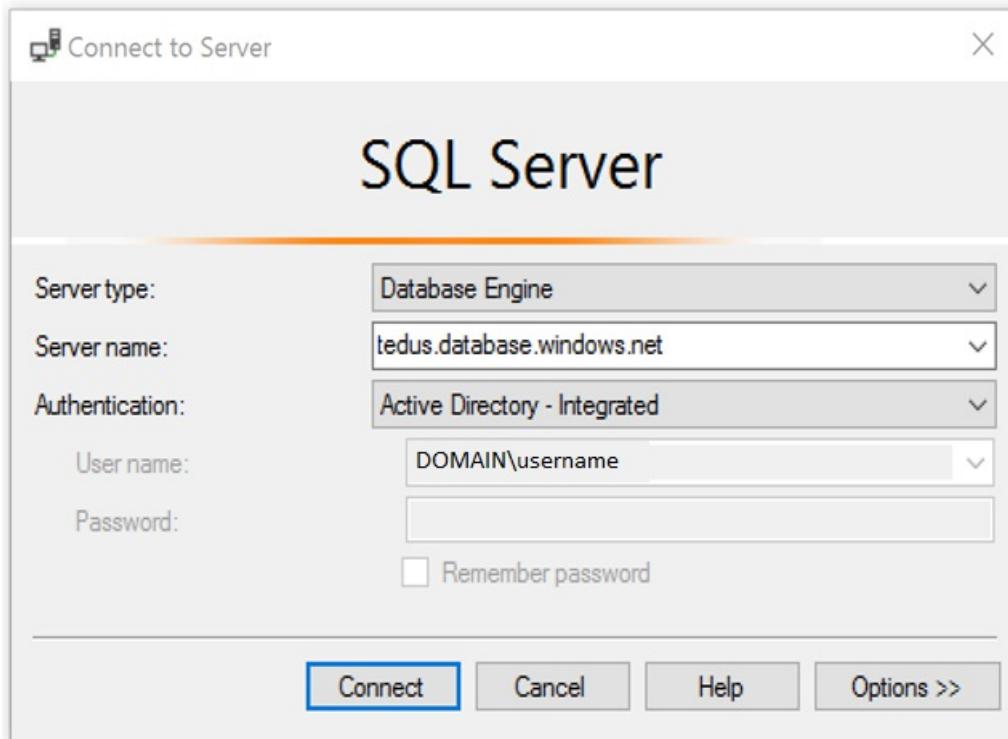
Using an Azure AD identity to connect using SSMS or SSDT

The following procedures show you how to connect to a SQL database with an Azure AD identity using SQL Server Management Studio or SQL Server Database Tools.

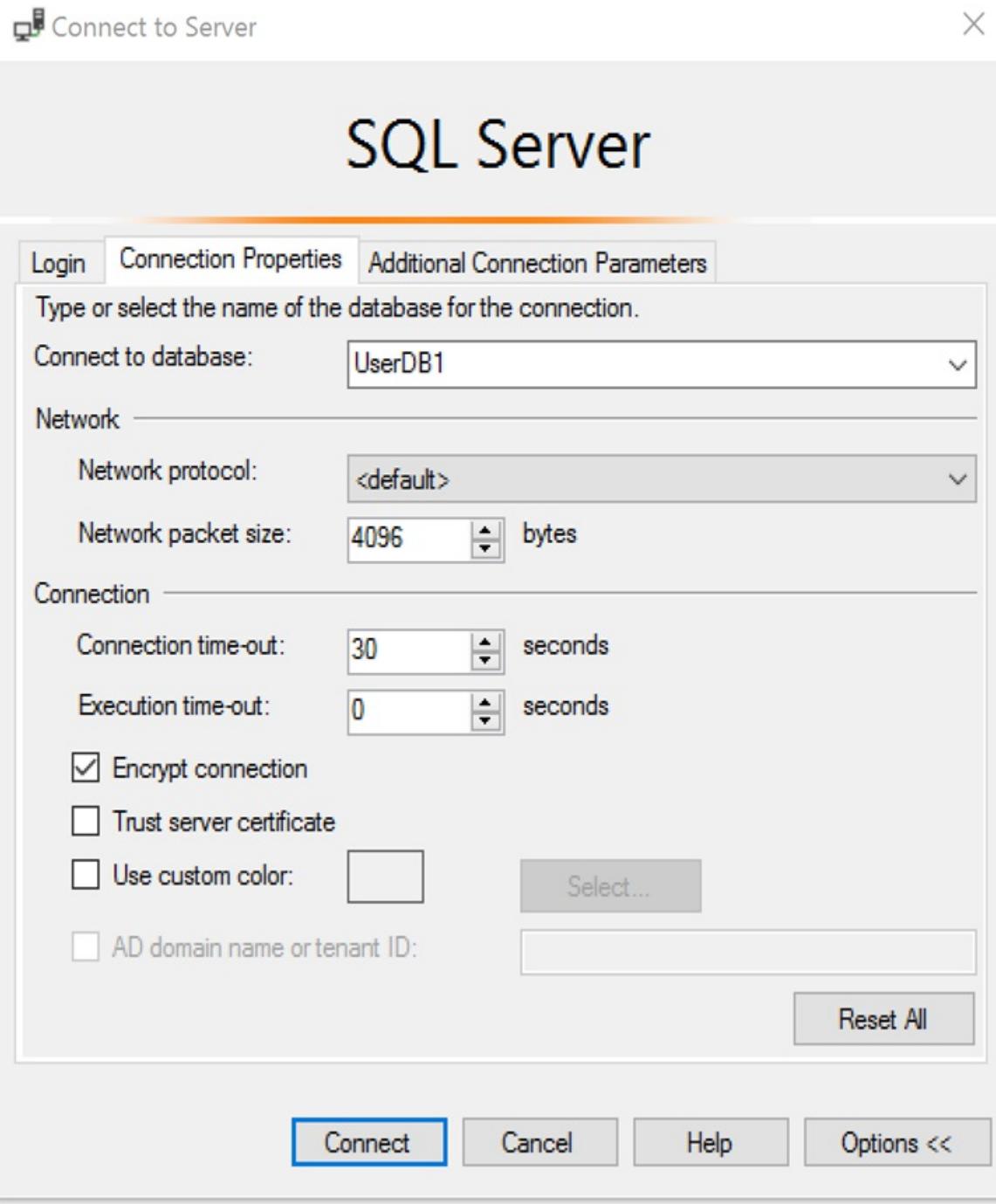
Active Directory integrated authentication

Use this method if you are logged in to Windows using your Azure Active Directory credentials from a federated domain.

1. Start Management Studio or Data Tools and in the **Connect to Server** (or **Connect to Database Engine**) dialog box, in the **Authentication** box, select **Active Directory - Integrated**. No password is needed or can be entered because your existing credentials will be presented for the connection.



2. Select the **Options** button, and on the **Connection Properties** page, in the **Connect to database** box, type the name of the user database you want to connect to. (The **AD domain name or tenant ID** option is only supported for **Universal with MFA connection** options, otherwise it is greyed out.)



Active Directory password authentication

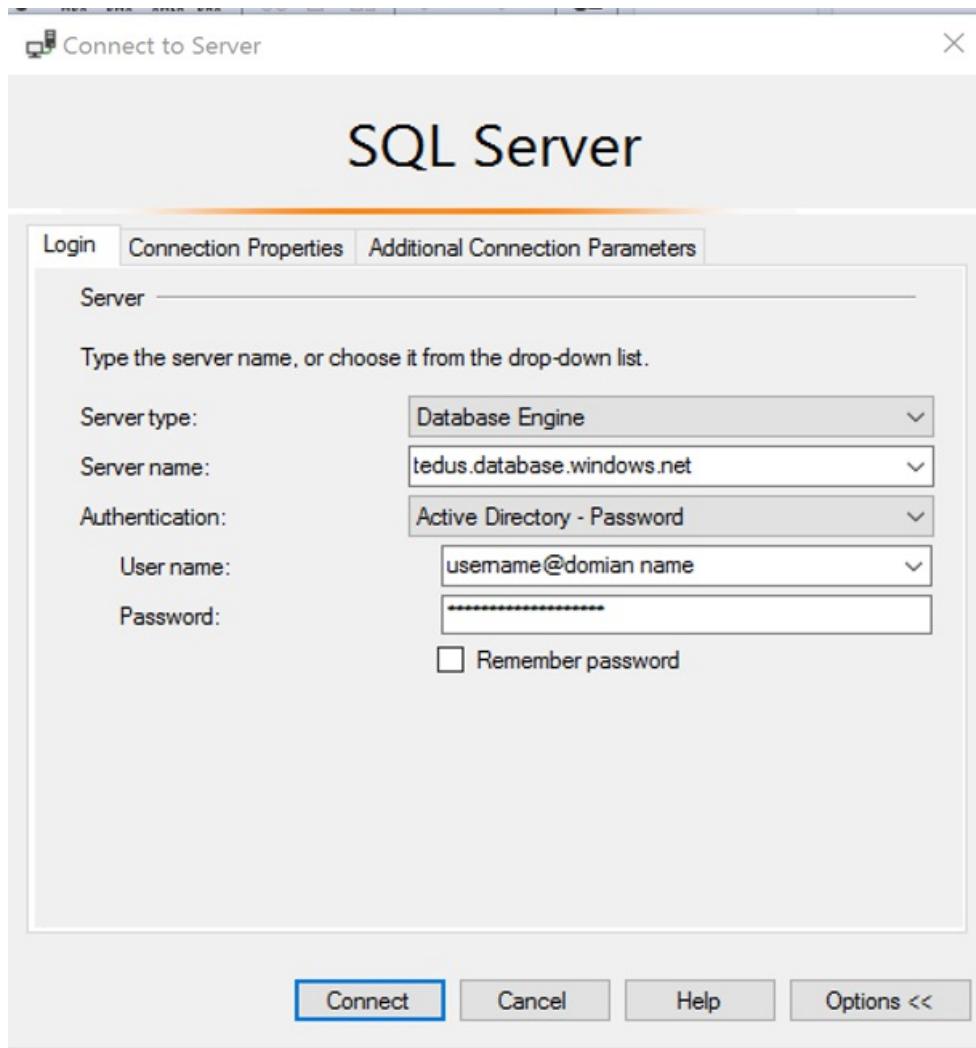
Use this method when connecting with an Azure AD principal name using the Azure AD managed domain. You can also use it for federated accounts without access to the domain, for example when working remotely.

Use this method to authenticate to SQL DB/DW with Azure AD for native or federated Azure AD users. A native user is one explicitly created in Azure AD and being authenticated using user name and password, while a federated user is a Windows user whose domain is federated with Azure AD. The latter method (using user & password) can be used when a user wants to use their windows credential, but their local machine is not joined with the domain (for example, using a remote access). In this case, a Windows user can indicate their domain account and password and can authenticate to SQL DB/DW using federated credentials.

1. Start Management Studio or Data Tools and in the **Connect to Server** (or **Connect to Database Engine**) dialog box, in the **Authentication** box, select **Active Directory - Password**.
2. In the **User name** box, type your Azure Active Directory user name in the format **username@domain.com**. User names must be an account from the Azure Active Directory or an

account from a domain federate with the Azure Active Directory.

3. In the **Password** box, type your user password for the Azure Active Directory account or federated domain account.



4. Select the **Options** button, and on the **Connection Properties** page, in the **Connect to database** box, type the name of the user database you want to connect to. (See the graphic in the previous option.)

Using an Azure AD identity to connect from a client application

The following procedures show you how to connect to a SQL database with an Azure AD identity from a client application.

Active Directory integrated authentication

To use integrated Windows authentication, your domain's Active Directory must be federated with Azure Active Directory. Your client application (or a service) connecting to the database must be running on a domain-joined machine under a user's domain credentials.

To connect to a database using integrated authentication and an Azure AD identity, the Authentication keyword in the database connection string must be set to Active Directory Integrated. The following C# code sample uses ADO .NET.

```
string ConnectionString = @"Data Source=n9lxnyuzhv.database.windows.net; Authentication=Active Directory Integrated; Initial Catalog=testdb;";
SqlConnection conn = new SqlConnection(ConnectionString);
conn.Open();
```

The connection string keyword `Integrated Security=True` is not supported for connecting to Azure SQL Database. When making an ODBC connection, you will need to remove spaces and set Authentication to 'ActiveDirectoryIntegrated'.

Active Directory password authentication

To connect to a database using integrated authentication and an Azure AD identity, the Authentication keyword must be set to Active Directory Password. The connection string must contain User ID/UID and Password/PWD keywords and values. The following C# code sample uses ADO .NET.

```
string ConnectionString =
@Data Source=n9lxnyuzhv.database.windows.net; Authentication=Active Directory Password; Initial
Catalog=testdb; UID=bob@contoso.onmicrosoft.com; PWD=MyPassWord!";
SqlConnection conn = new SqlConnection(ConnectionString);
conn.Open();
```

Learn more about Azure AD authentication methods using the demo code samples available at [Azure AD Authentication GitHub Demo](#).

Azure AD token

This authentication method allows middle-tier services to connect to Azure SQL Database or Azure SQL Data Warehouse by obtaining a token from Azure Active Directory (AAD). It enables sophisticated scenarios including certificate-based authentication. You must complete four basic steps to use Azure AD token authentication:

1. Register your application with Azure Active Directory and get the client ID for your code.
2. Create a database user representing the application. (Completed earlier in step 6.)
3. Create a certificate on the client computer runs the application.
4. Add the certificate as a key for your application.

Sample connection string:

```
string ConnectionString =@"Data Source=n9lxnyuzhv.database.windows.net; Initial Catalog=testdb;";
SqlConnection conn = new SqlConnection(ConnectionString);
conn.AccessToken = "Your JWT token"
conn.Open();
```

For more information, see [SQL Server Security Blog](#). For information about adding a certificate, see [Get started with certificate-based authentication in Azure Active Directory](#).

sqlcmd

The following statements, connect using version 13.1 of sqlcmd, which is available from the [Download Center](#).

NOTE

`sqlcmd` with the `-G` command does not work with system identities, and requires a user principal login.

```
sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net -G
sqlcmd -S Target_DB_or_DW.testsrv.database.windows.net -U bob@contoso.com -P MyAADPassword -G -l 30
```

Troubleshooting Azure AD Authentication

Guidance on troubleshooting issues with Azure AD Authentication can be found in the following blog:
[https://techcommunity.microsoft.com/t5/azure-sql-database/troubleshooting-problems-related-to-azure-ad-](https://techcommunity.microsoft.com/t5/azure-sql-database/troubleshooting-problems-related-to-azure-ad/)

Next steps

- For an overview of access and control in SQL Database, see [SQL Database access and control](#).
- For an overview of logins, users, and database roles in SQL Database, see [Logins, users, and database roles](#).
- For more information about database principals, see [Principals](#).
- For more information about database roles, see [Database roles](#).
- For more information about firewall rules in SQL Database, see [SQL Database firewall rules](#).

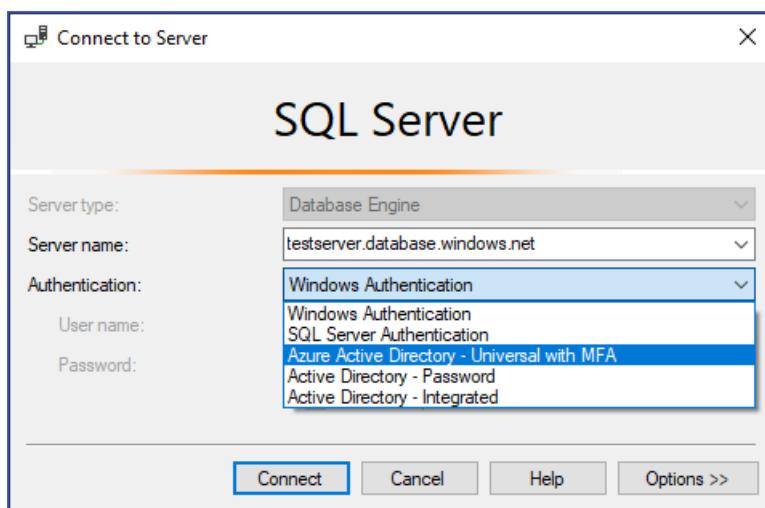
Using Multi-factor AAD authentication with Azure SQL Database and Azure SQL Data Warehouse (SSMS support for MFA)

11/7/2019 • 5 minutes to read • [Edit Online](#)

Azure SQL Database and Azure SQL Data Warehouse support connections from SQL Server Management Studio (SSMS) using *Active Directory Universal Authentication*. This article discusses the differences between the various authentication options, and also the limitations associated with using Universal Authentication.

Download the latest SSMS - On the client computer, download the latest version of SSMS, from [Download SQL Server Management Studio \(SSMS\)](#).

For all the features discussed in this article, use at least July 2017, version 17.2. The most recent connection dialog box, should look similar to the following image:



The five authentication options

Active Directory Universal Authentication supports the two non-interactive authentication methods: -

`Active Directory - Password` authentication - `Active Directory - Integrated` authentication

There are two non-interactive authentication models as well, which can be used in many different applications (ADO.NET, JDBC, ODC, etc.). These two methods never result in pop-up dialog boxes:

- `Active Directory - Password`
- `Active Directory - Integrated`

The interactive method is that also supports Azure multi-factor authentication (MFA) is:

- `Active Directory - Universal with MFA`

Azure MFA helps safeguard access to data and applications while meeting user demand for a simple sign-in process. It delivers strong authentication with a range of easy verification options (phone call, text message, smart cards with pin, or mobile app notification), allowing users to choose the method they prefer. Interactive MFA with Azure AD can result in a pop-up dialog box for validation.

For a description of Multi-Factor Authentication, see [Multi-Factor Authentication](#). For configuration steps, see [Configure Azure SQL Database multi-factor authentication for SQL Server Management Studio](#).

Azure AD domain name or tenant ID parameter

Beginning with [SSMS version 17](#), users that are imported into the current Active Directory from other Azure Active Directories as guest users, can provide the Azure AD domain name, or tenant ID when they connect. Guest users include users invited from other Azure ADs, Microsoft accounts such as outlook.com, hotmail.com, live.com, or other accounts like gmail.com. This information, allows **Active Directory Universal with MFA Authentication** to identify the correct authenticating authority. This option is also required to support Microsoft accounts (MSA) such as outlook.com, hotmail.com, live.com, or non-MSA accounts. All these users who want to be authenticated using Universal Authentication must enter their Azure AD domain name or tenant ID. This parameter represents the current Azure AD domain name/tenant ID the Azure Server is linked with. For example, if Azure Server is associated with Azure AD domain `contosotest.onmicrosoft.com` where user `joe@contosodev.onmicrosoft.com` is hosted as an imported user from Azure AD domain `contosodev.onmicrosoft.com`, the domain name required to authenticate this user is `contosotest.onmicrosoft.com`. When the user is a native user of the Azure AD linked to Azure Server, and is not an MSA account, no domain name or tenant ID is required. To enter the parameter (beginning with SSMS version 17.2), in the **Connect to Database** dialog box, complete the dialog box, selecting **Active Directory - Universal with MFA** authentication, click **Options**, complete the **User name** box, and then click the **Connection Properties** tab. Check the **AD domain name or tenant ID** box, and provide authenticating authority, such as the domain name (**contosotest.onmicrosoft.com**) or the GUID of the tenant ID.

SQL Server

SQL Server

Connection Properties

Type or select the name of the database for the connection.

Connect to database: MyDatabase

Network

Network protocol: <default>

Network packet size: 4096 bytes

Connection

Connection time-out: 30 seconds

Execution time-out: 0 seconds

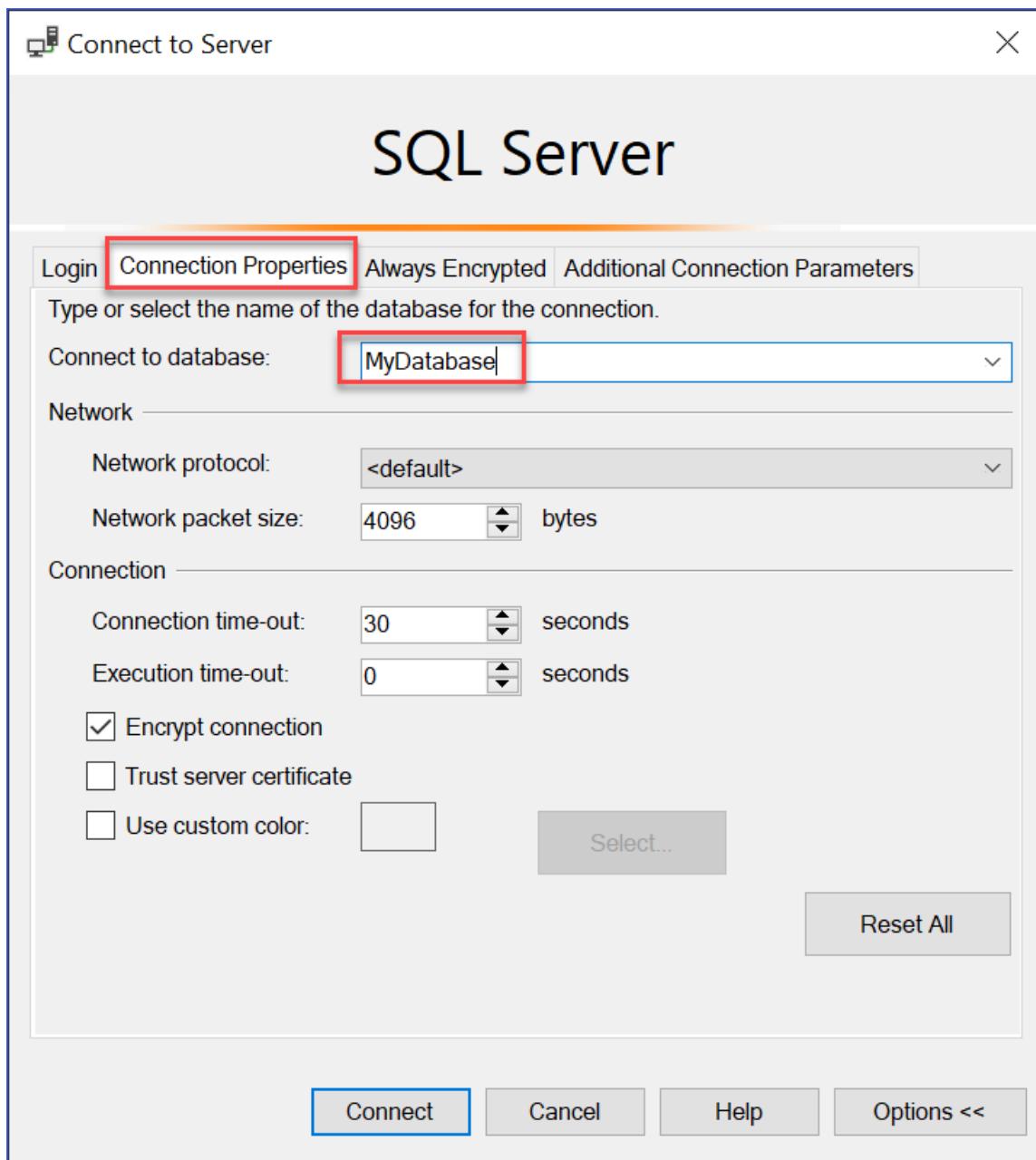
Encrypt connection

Trust server certificate

Use custom color:

AD domain name or tenant ID: contosotest.onmicrosoft.com

If you are running SSMS 18.x or later then the AD domain name or tenant ID is no longer needed for guest users because 18.x or later automatically recognizes it.



Azure AD business to business support

Azure AD users supported for Azure AD B2B scenarios as guest users (see [What is Azure B2B collaboration](#)) can connect to SQL Database and SQL Data Warehouse only as part of members of a group created in current Azure AD and mapped manually using the Transact-SQL `CREATE USER` statement in a given database. For example, if `steve@gmail.com` is invited to Azure AD `contosotest` (with the Azure Ad domain `contosotest.onmicrosoft.com`), an Azure AD group, such as `usergroup` must be created in the Azure AD that contains the `steve@gmail.com` member. Then, this group must be created for a specific database (that is, `MyDatabase`) by Azure AD SQL admin or Azure AD DBO by executing a Transact-SQL `CREATE USER [usergroup] FROM EXTERNAL PROVIDER` statement. After the database user is created, then the user `steve@gmail.com` can log in to `MyDatabase` using the SSMS authentication option `Active Directory - Universal with MFA support`. The usergroup, by default, has only the connect permission and any further data access that will need to be granted in the normal way. Note that user `steve@gmail.com` as a guest user must check the box and add the AD domain name `contosotest.onmicrosoft.com` in the SSMS **Connection Property** dialog box. The **AD domain name or tenant ID** option is only supported for the Universal with MFA connection options, otherwise it is greyed out.

Universal Authentication limitations for SQL Database and SQL Data Warehouse

- SSMS and SqlPackage.exe are the only tools currently enabled for MFA through Active Directory Universal Authentication.
- SSMS version 17.2, supports multi-user concurrent access using Universal Authentication with MFA. Version 17.0 and 17.1, restricted a login for an instance of SSMS using Universal Authentication to a single Azure Active Directory account. To log in as another Azure AD account, you must use another instance of SSMS. (This restriction is limited to Active Directory Universal Authentication; you can log in to different servers using Active Directory Password Authentication, Active Directory Integrated Authentication, or SQL Server Authentication).
- SSMS supports Active Directory Universal Authentication for Object Explorer, Query Editor, and Query Store visualization.
- SSMS version 17.2 provides DacFx Wizard support for Export/Extract/Deploy Data database. Once a specific user is authenticated through the initial authentication dialog using Universal Authentication, the DacFx Wizard functions the same way it does for all other authentication methods.
- The SSMS Table Designer does not support Universal Authentication.
- There are no additional software requirements for Active Directory Universal Authentication except that you must use a supported version of SSMS.
- The Active Directory Authentication Library (ADAL) version for Universal authentication was updated to its latest ADAL.dll 3.13.9 available released version. See [Active Directory Authentication Library 3.14.1](#).

Next steps

- For configuration steps, see [Configure Azure SQL Database multi-factor authentication for SQL Server Management Studio](#).
- Grant others access to your database: [SQL Database Authentication and Authorization: Granting Access](#)
- Make sure others can connect through the firewall: [Configure an Azure SQL Database server-level firewall rule using the Azure portal](#)
- [Configure and manage Azure Active Directory authentication with SQL Database or SQL Data Warehouse](#)
- [Microsoft SQL Server Data-Tier Application Framework \(17.0.0 GA\)](#)
- [SQLPackage.exe](#)
- [Import a BACPAC file to a new Azure SQL Database](#)
- [Export an Azure SQL database to a BACPAC file](#)
- [C# interface IUniversalAuthProvider Interface](#)
- When using **Active Directory- Universal with MFA** authentication, ADAL tracing is available beginning with [SSMS 17.3](#). Off by default, you can turn on ADAL tracing by using the **Tools, Options** menu, under **Azure Services, Azure Cloud, ADAL Output Window Trace Level**, followed by enabling **Output** in the **View** menu. The traces are available in the output window when selecting **Azure Active Directory option**.

Configure multi-factor authentication for SQL Server Management Studio and Azure AD

11/7/2019 • 3 minutes to read • [Edit Online](#)

This topic shows you how to use Azure Active Directory multi-factor authentication (MFA) with SQL Server Management Studio. Azure AD MFA can be used when connecting SSMS or SqlPackage.exe to Azure [SQL Database](#) and [SQL Data Warehouse](#). For an overview of Azure SQL Database multi-factor authentication, see [Universal Authentication with SQL Database and SQL Data Warehouse \(SSMS support for MFA\)](#).

NOTE

This topic applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

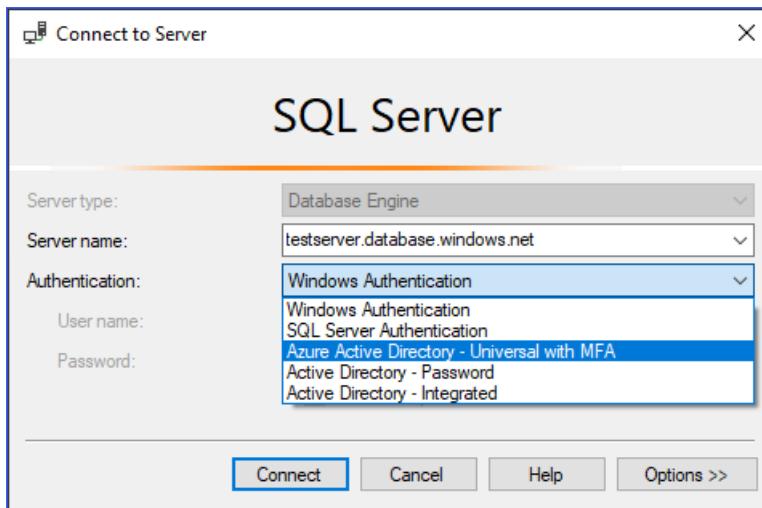
Configuration steps

1. **Configure an Azure Active Directory** - For more information, see [Administering your Azure AD directory](#), [Integrating your on-premises identities with Azure Active Directory](#), [Add your own domain name to Azure AD](#), [Microsoft Azure now supports federation with Windows Server Active Directory](#), and [Manage Azure AD using Windows PowerShell](#).
2. **Configure MFA** - For step-by-step instructions, see [What is Azure Multi-Factor Authentication?](#), [Conditional Access \(MFA\) with Azure SQL Database and Data Warehouse](#). (Full Conditional Access requires a Premium Azure Active Directory (Azure AD). Limited MFA is available with a standard Azure AD.)
3. **Configure SQL Database or SQL Data Warehouse for Azure AD Authentication** - For step-by-step instructions, see [Connecting to SQL Database or SQL Data Warehouse By Using Azure Active Directory Authentication](#).
4. **Download SSMS** - On the client computer, download the latest SSMS, from [Download SQL Server Management Studio \(SSMS\)](#). For all the features in this topic, use at least July 2017, version 17.2.

Connecting by using universal authentication with SSMS

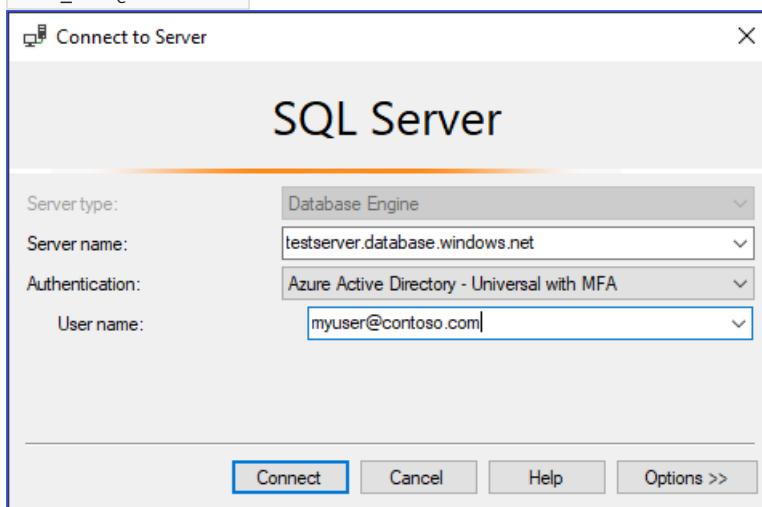
The following steps show how to connect to SQL Database or SQL Data Warehouse by using the latest SSMS.

1. To connect using Universal Authentication, on the **Connect to Server** dialog box, select **Active Directory - Universal with MFA support**. (If you see **Active Directory Universal Authentication** you are not on the latest version of SSMS.)

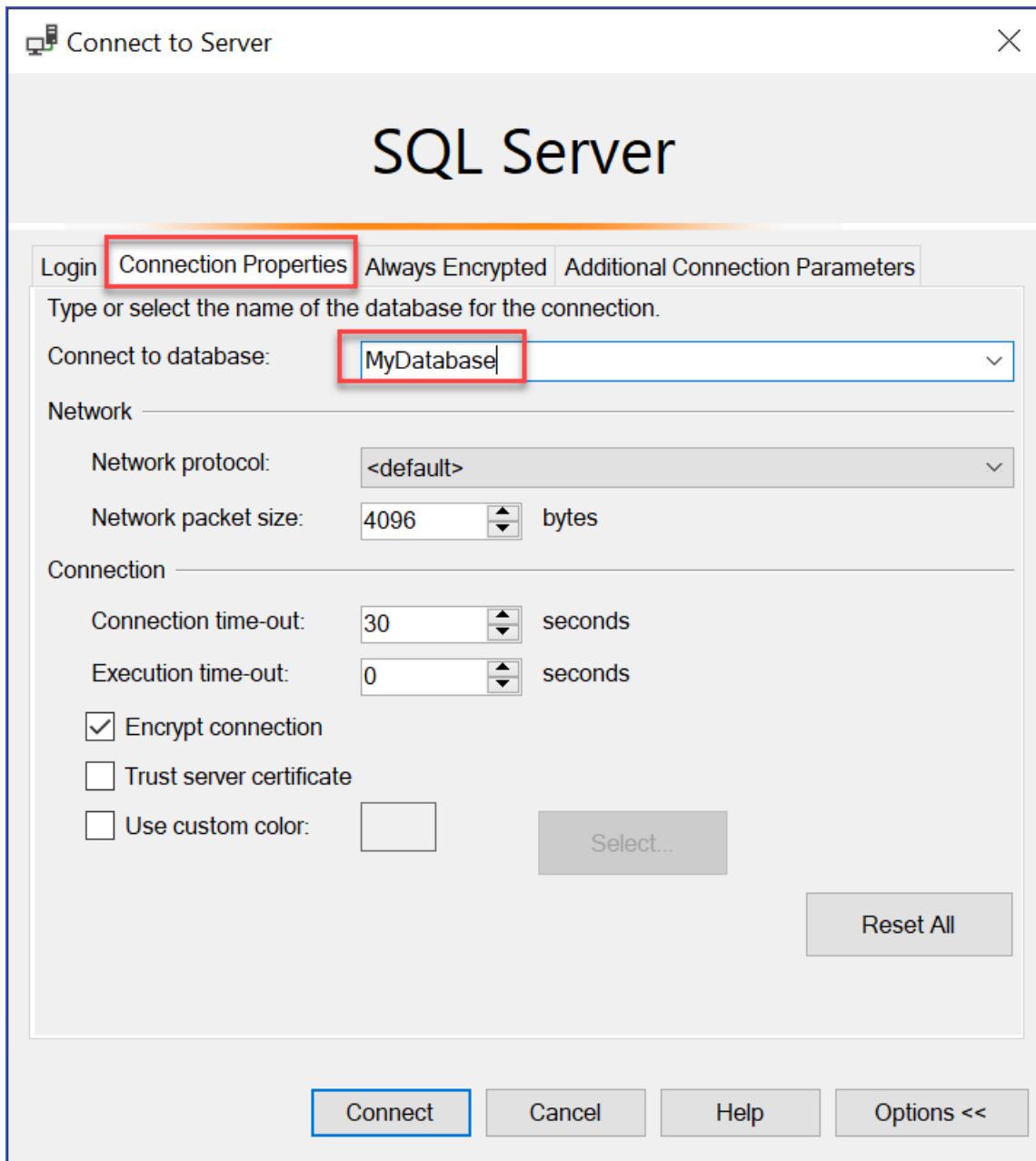


2. Complete the **User name** box with the Azure Active Directory credentials, in the format

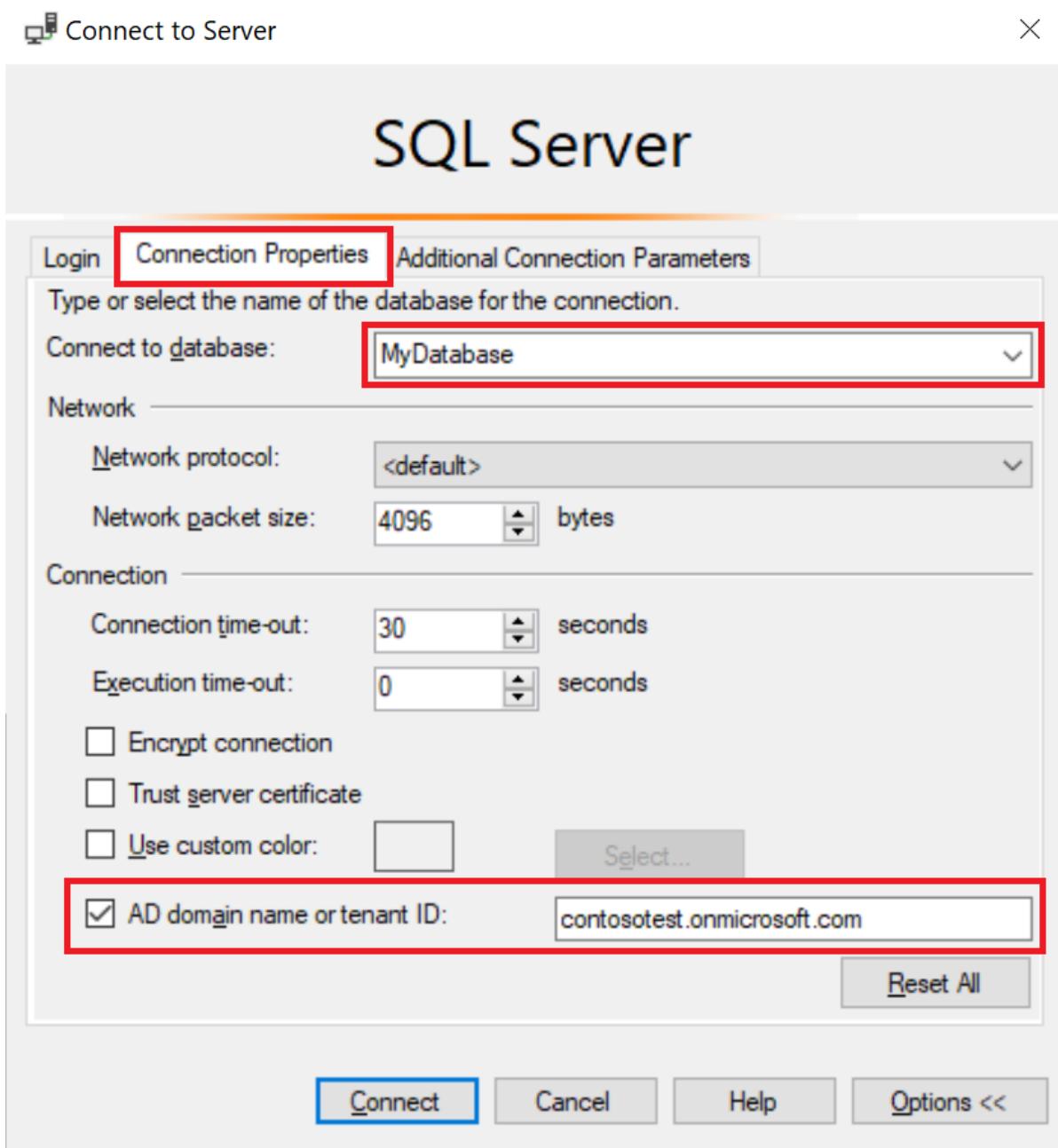
user_name@domain.com .



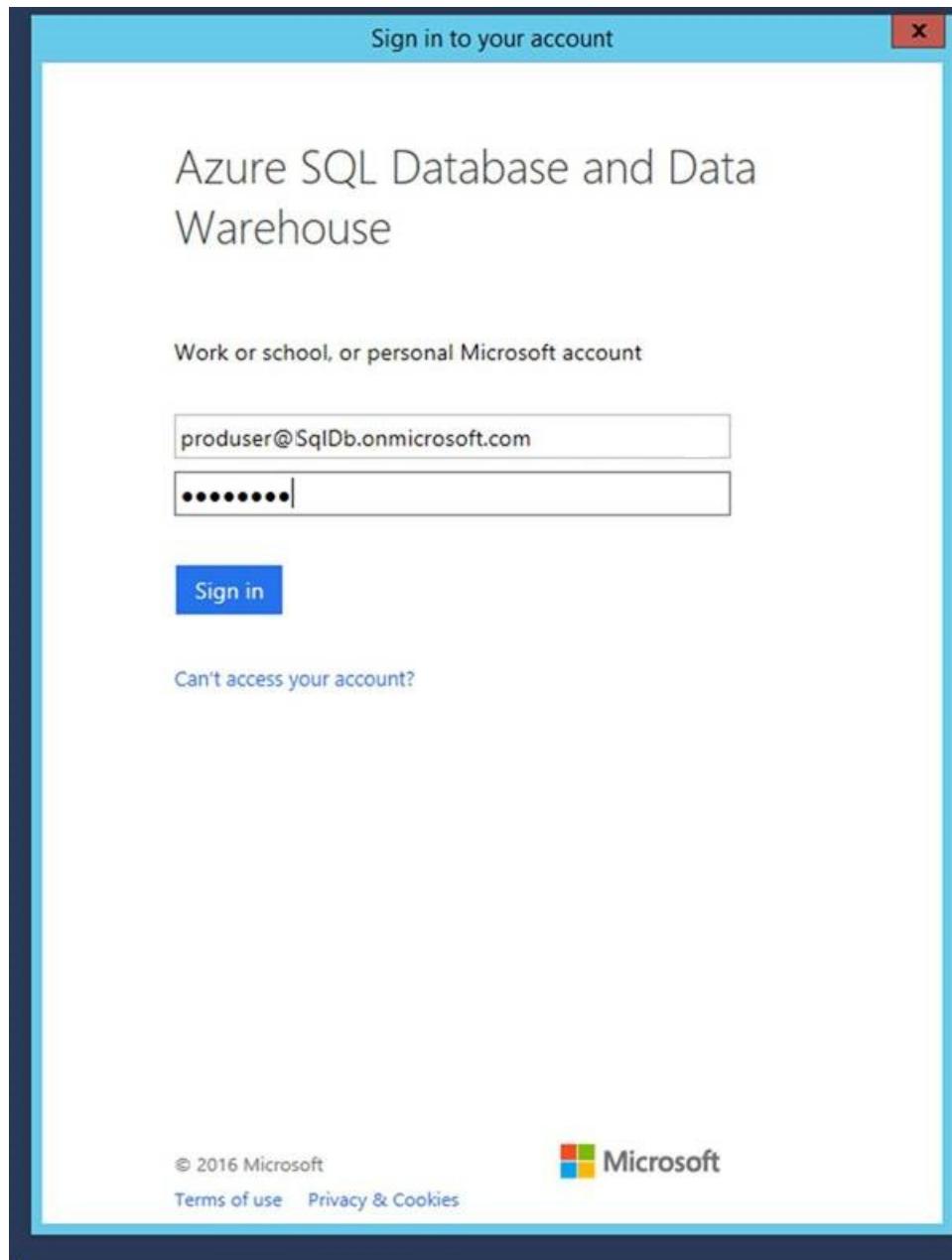
3. If you are connecting as a guest user, you no longer need to complete the AD domain name or tenant ID field for guest users because SSMS 18.x or later automatically recognizes it. For more information, see [Universal Authentication with SQL Database and SQL Data Warehouse \(SSMS support for MFA\)](#).



However, If you are connecting as a guest user using SSMS 17.x or older, you must click **Options**, and on the **Connection Property** dialog box, and complete the **AD domain name or tenant ID** box.



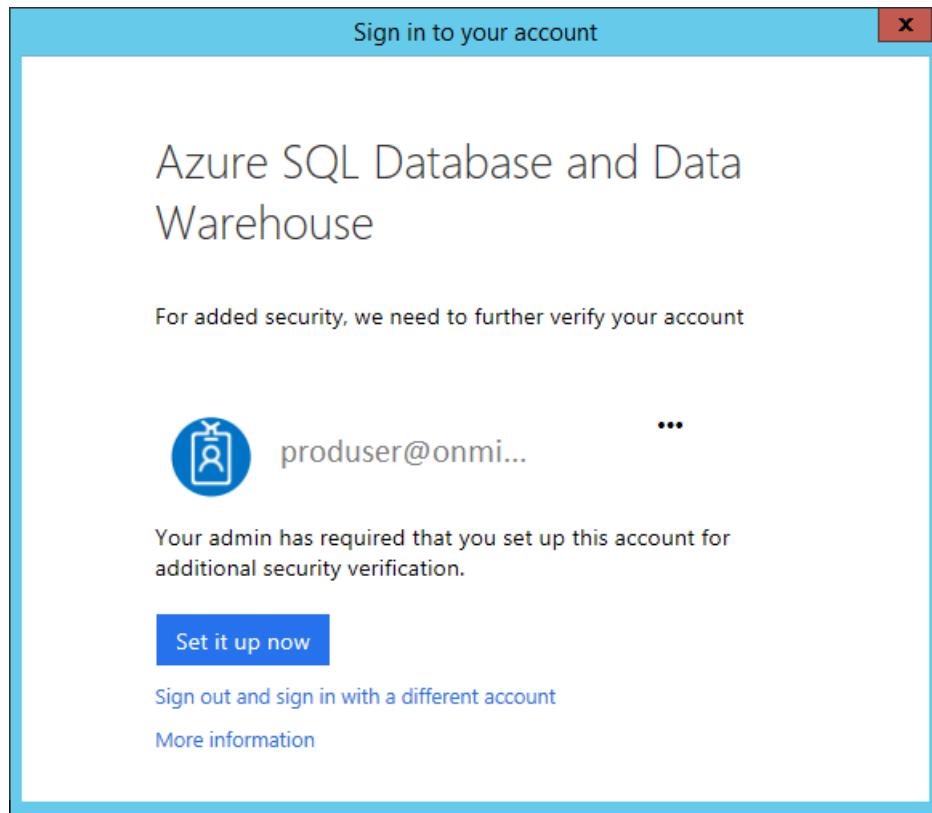
4. As usual for SQL Database and SQL Data Warehouse, you must click **Options** and specify the database on the **Options** dialog box. (If the connected user is a guest user (i.e. joe@outlook.com), you must check the box and add the current AD domain name or tenant ID as part of Options. See [Universal Authentication with SQL Database and SQL Data Warehouse \(SSMS support for MFA\)](#). Then click **Connect**.
5. When the **Sign in to your account** dialog box appears, provide the account and password of your Azure Active Directory identity. No password is required if a user is part of a domain federated with Azure AD.



NOTE

For Universal Authentication with an account that does not require MFA, you connect at this point. For users requiring MFA, continue with the following steps:

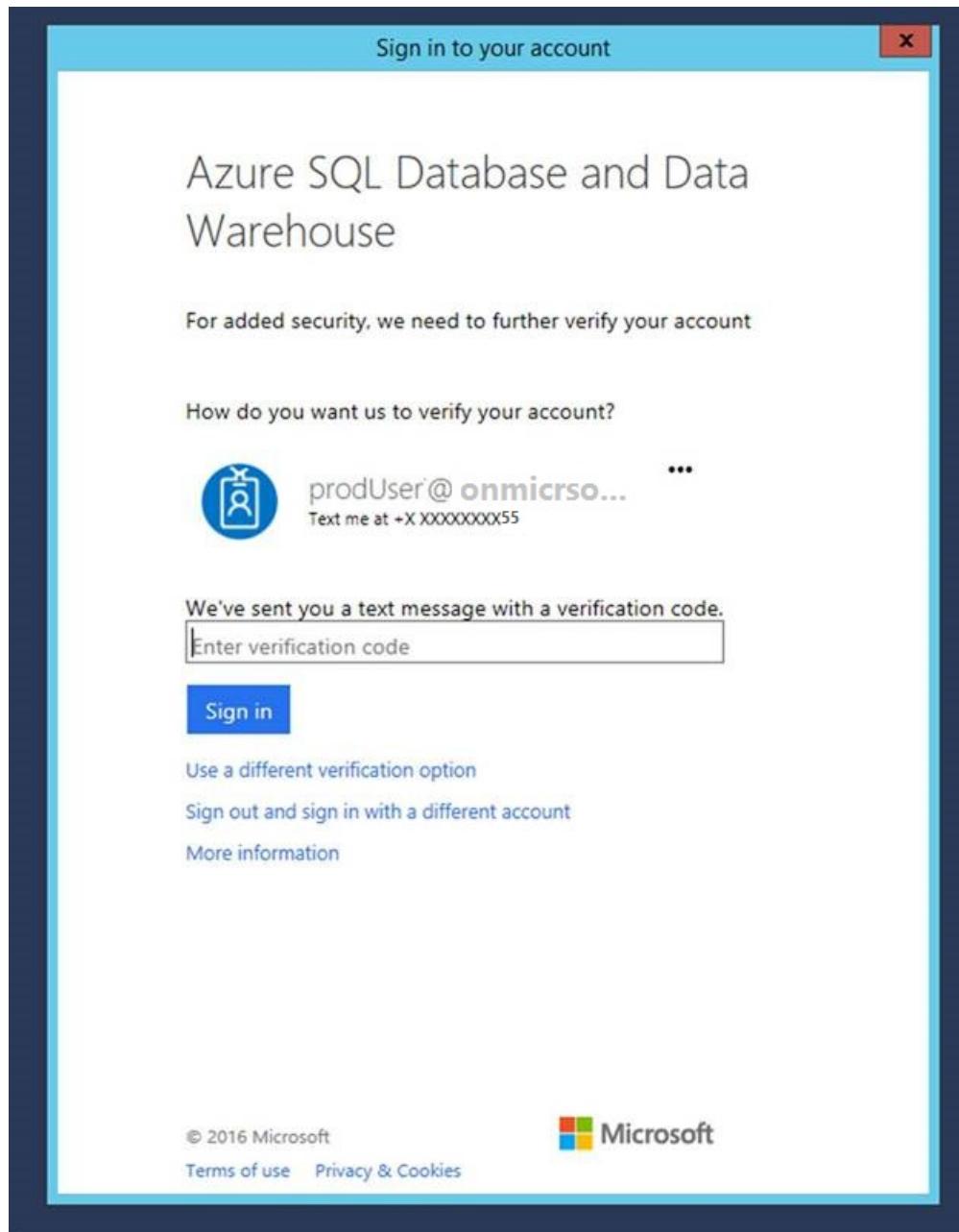
6. Two MFA setup dialog boxes might appear. This one time operation depends on the MFA administrator setting, and therefore may be optional. For an MFA enabled domain this step is sometimes pre-defined (for example, the domain requires users to use a smartcard and pin).



7. The second possible one time dialog box allows you to select the details of your authentication method. The possible options are configured by your administrator.

The dialog box has a blue header bar with the text "Additional security verification" and a red close button. The main content area has a light gray background. At the top, it says "Microsoft Azure". Below that, a large heading says "Additional security verification". A sub-instruction says "Secure your account by adding phone verification to your password. [View video](#)". A section titled "Step 1: How should we contact you?" contains a dropdown menu set to "Authentication phone", a dropdown menu set to "United States (+1)" with the number "18005555555" entered, and a "Method" section with two radio buttons: "Send me a code by text message" (which is selected) and "Call me". A blue "Contact me" button is at the bottom. A note at the bottom states: "Your phone numbers will only be used for account security. Standard telephone and SMS charges will apply."

8. The Azure Active Directory sends the confirming information to you. When you receive the verification code, enter it into the **Enter verification code** box, and click **Sign in**.



When verification is complete, SSMS connects normally presuming valid credentials and firewall access.

Next steps

- For an overview of Azure SQL Database multi-factor authentication, see [Universal Authentication with SQL Database and SQL Data Warehouse \(SSMS support for MFA\)](#).
- Grant others access to your database: [SQL Database Authentication and Authorization: Granting Access](#)
- Make sure others can connect through the firewall: [Configure an Azure SQL Database server-level firewall rule using the Azure portal](#)
- When using **Active Directory- Universal with MFA** authentication, ADAL tracing is available beginning with [SSMS 17.3](#). Off by default, you can turn on ADAL tracing by using the **Tools, Options** menu, under **Azure Services, Azure Cloud, ADAL Output Window Trace Level**, followed by enabling **Output** in the **View** menu. The traces are available in the output window when selecting **Azure Active Directory option**.

Transparent data encryption for SQL Database and Data Warehouse

1/14/2020 • 10 minutes to read • [Edit Online](#)

Transparent data encryption (TDE) helps protect Azure SQL Database, Azure SQL Managed Instance, and Azure Data Warehouse against the threat of malicious offline activity by encrypting data at rest. It performs real-time encryption and decryption of the database, associated backups, and transaction log files at rest without requiring changes to the application. By default, TDE is enabled for all newly deployed Azure SQL databases. TDE cannot be used to encrypt the logical **master** database in SQL Database. The **master** database contains objects that are needed to perform the TDE operations on the user databases.

TDE needs to be manually enabled for older databases of Azure SQL Database, Azure SQL Managed Instance, or Azure SQL Data Warehouse. Managed Instance databases created through restore inherit encryption status from the source database.

Transparent data encryption encrypts the storage of an entire database by using a symmetric key called the database encryption key. This database encryption key is protected by the transparent data encryption protector. The protector is either a service-managed certificate (service-managed transparent data encryption) or an asymmetric key stored in Azure Key Vault (Bring Your Own Key). You set the transparent data encryption protector at the server level for Azure SQL Database and Data Warehouse, and instance level for Azure SQL Managed Instance. The term *server* refers both to server and instance throughout this document, unless stated differently.

On database startup, the encrypted database encryption key is decrypted and then used for decryption and re-encryption of the database files in the SQL Server Database Engine process. Transparent data encryption performs real-time I/O encryption and decryption of the data at the page level. Each page is decrypted when it's read into memory and then encrypted before being written to disk. For a general description of transparent data encryption, see [Transparent data encryption](#).

SQL Server running on an Azure virtual machine also can use an asymmetric key from Key Vault. The configuration steps are different from using an asymmetric key in SQL Database and SQL Managed Instance. For more information, see [Extensible key management by using Azure Key Vault \(SQL Server\)](#).

Service-managed transparent data encryption

In Azure, the default setting for transparent data encryption is that the database encryption key is protected by a built-in server certificate. The built-in server certificate is unique for each server and the encryption algorithm used is AES 256. If a database is in a geo-replication relationship, both the primary and geo-secondary database are protected by the primary database's parent server key. If two databases are connected to the same server, they also share the same built-in certificate. Microsoft automatically rotates these certificates in compliance with the internal security policy and the root key is protected by a Microsoft internal secret store. Customers can verify SQL Database compliance with internal security policies in independent third-party audit reports available on the [Microsoft Trust Center](#).

Microsoft also seamlessly moves and manages the keys as needed for geo-replication and restores.

IMPORTANT

All newly created SQL databases and Managed Instance databases are encrypted by default by using service-managed transparent data encryption. Existing SQL databases created before May 2017 and SQL databases created through restore, geo-replication, and database copy are not encrypted by default. Existing Managed Instance databases created before February 2019 are not encrypted by default. Managed Instance databases created through restore inherit encryption status from the source.

Customer-managed transparent data encryption - Bring Your Own Key

TDE with customer-managed keys in Azure Key Vault allows to encrypt the Database Encryption Key (DEK) with a customer-managed asymmetric key called TDE Protector. This is also generally referred to as Bring Your Own Key (BYOK) support for Transparent Data Encryption. In the BYOK scenario, the TDE Protector is stored in a customer-owned and managed [Azure Key Vault](#), Azure's cloud-based external key management system. The TDE Protector can be [generated by the key vault or transferred to the key vault](#) from an on premises HSM device. The TDE DEK, which is stored on the boot page of a database, is encrypted and decrypted by the TDE Protector, which is stored in Azure Key Vault and never leaves the key vault. SQL Database needs to be granted permissions to the customer-owned key vault to decrypt and encrypt the DEK. If permissions of the logical SQL server to the key vault are revoked, a database will be inaccessible and all data is encrypted. For Azure SQL Database, the TDE protector is set at the logical SQL server level and is inherited by all databases associated with that server. For [Azure SQL Managed Instance](#), the TDE protector is set at the instance level and it is inherited by all *encrypted* databases on that instance. The term *server* refers both to server and instance throughout this document, unless stated differently.

With TDE with Azure Key Vault integration, users can control key management tasks including key rotations, key vault permissions, key backups, and enable auditing/reporting on all TDE protectors using Azure Key Vault functionality. Key Vault provides central key management, leverages tightly monitored hardware security modules (HSMs), and enables separation of duties between management of keys and data to help meet compliance with security policies. To learn more about transparent data encryption with Azure Key Vault integration (Bring Your Own Key support) for Azure SQL Database, SQL Managed Instance, and Data Warehouse, see [Transparent data encryption with Azure Key Vault integration](#).

To start using transparent data encryption with Azure Key Vault integration (Bring Your Own Key support), see the how-to guide [Turn on transparent data encryption by using your own key from Key Vault by using PowerShell](#).

Move a transparent data encryption-protected database

You don't need to decrypt databases for operations within Azure. The transparent data encryption settings on the source database or primary database are transparently inherited on the target. Operations that are included involve:

- Geo-restore
- Self-service point-in-time restore
- Restoration of a deleted database
- Active geo-replication
- Creation of a database copy
- Restore of backup file to Azure SQL Managed Instance

IMPORTANT

Taking manual COPY-ONLY backup of a database encrypted by service-managed TDE is not allowed in Azure SQL Managed Instance, since certificate used for encryption is not accessible. Use point-in-time-restore feature to move this type of database to another Managed Instance.

When you export a transparent data encryption-protected database, the exported content of the database isn't encrypted. This exported content is stored in un-encrypted BACPAC files. Be sure to protect the BACPAC files appropriately and enable transparent data encryption after import of the new database is finished.

For example, if the BACPAC file is exported from an on-premises SQL Server instance, the imported content of the new database isn't automatically encrypted. Likewise, if the BACPAC file is exported to an on-premises SQL Server instance, the new database also isn't automatically encrypted.

The one exception is when you export to and from a SQL database. Transparent data encryption is enabled in the new database, but the BACPAC file itself still isn't encrypted.

Manage transparent data encryption

- [Portal](#)
- [PowerShell](#)
- [Transact-SQL](#)
- [REST API](#)

Manage transparent data encryption in the Azure portal.

To configure transparent data encryption through the Azure portal, you must be connected as the Azure Owner, Contributor, or SQL Security Manager.

You turn transparent data encryption on and off on the database level. To enable transparent data encryption on a database, go to the [Azure portal](#) and sign in with your Azure Administrator or Contributor account. Find the transparent data encryption settings under your user database. By default, service-managed transparent data encryption is used. A transparent data encryption certificate is automatically generated for the server that contains the database. For Azure SQL Managed Instance use T-SQL to turn transparent data encryption on and off on a database.

The screenshot shows the Azure portal interface for managing database settings. The left sidebar has sections for Home, SQL databases, contoso-database (contoso-sql-server/contoso-database), and Transparent data encryption. The main content area is titled 'contoso-database (contoso-sql-server/contoso-database) - Transparent data encryption'. It includes a search bar, save, discard, and feedback buttons. A 'Learn more' link provides information about TDE. The 'Data encryption' section has a 'ON' button, which is highlighted in blue. Below it, the 'Encryption status' is shown as 'Encrypted' with a green checkmark. The sidebar also lists other database features like Export template, Stream analytics (preview), Advanced data security, Auditing, Dynamic Data Masking, and Intelligent Performance.

You set the transparent data encryption master key, also known as the transparent data encryption protector, on the server level. To use transparent data encryption with Bring Your Own Key support and protect your databases with a key from Key Vault, open the transparent data encryption settings under your server.

The screenshot shows the Azure portal interface for managing a SQL managed instance named 'SampleServer'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Quick start, Connection strings, Active Directory admin, Pricing tier, Locks, Automation script), Security (Advanced Threat Protection, Transparent data encryption - which is selected and highlighted in blue), Monitoring, Diagnostic settings, and Support + troubleshooting.

The main content area displays the 'Transparent data encryption' settings. It includes a brief description of TDE: 'Encrypts your databases, backups, and logs at rest without any changes to your application.' and a link to enable TDE for each database. Below this, there's a section for 'Use your own key' with 'Yes' and 'No' buttons. A radio button is selected for 'Select a key', and the 'sqldbvault' key vault is listed. Under the 'Key' section, 'Select a key' is highlighted with a blue background. A checkbox is present to 'Make the selected key the default TDE protector'. A note explains that SQL uses Get, Wrap Key, Unwrap Key permissions to access the selected key vault, and if needed, it will try granting these permissions on behalf of the user.

Key Picker
Pick a key

+ Create a new key

JTest
RSA 2048

Azure SQL Transparent Data Encryption with customer-managed key

2/13/2020 • 13 minutes to read • [Edit Online](#)

Azure SQL [Transparent Data Encryption \(TDE\)](#) with customer-managed key enables Bring Your Own Key (BYOK) scenario for data protection at rest, and allows organizations to implement separation of duties in the management of keys and data. With customer-managed transparent data encryption, customer is responsible for and in full control of a key lifecycle management (key creation, upload, rotation, deletion), key usage permissions, and auditing of operations on keys.

In this scenario, the key used for encryption of the Database Encryption Key (DEK), called TDE protector, is a customer-managed asymmetric key stored in a customer-owned and customer-managed [Azure Key Vault \(AKV\)](#), a cloud-based external key management system. Key Vault is highly available and scalable secure storage for RSA cryptographic keys, optionally backed by FIPS 140-2 Level 2 validated hardware security modules (HSMs). It doesn't allow direct access to a stored key, but provides services of encryption/decryption using the key to the authorized entities. The key can be generated by the key vault, imported, or [transferred to the key vault from an on-prem HSM device](#).

For Azure SQL Database and Azure SQL Data Warehouse, the TDE protector is set at the logical server level and is inherited by all encrypted databases associated with that server. For Azure SQL Managed Instance, the TDE protector is set at the instance level and is inherited by all encrypted databases on that instance. The term *server* refers both to SQL Database logical server and managed instance throughout this document, unless stated differently.

IMPORTANT

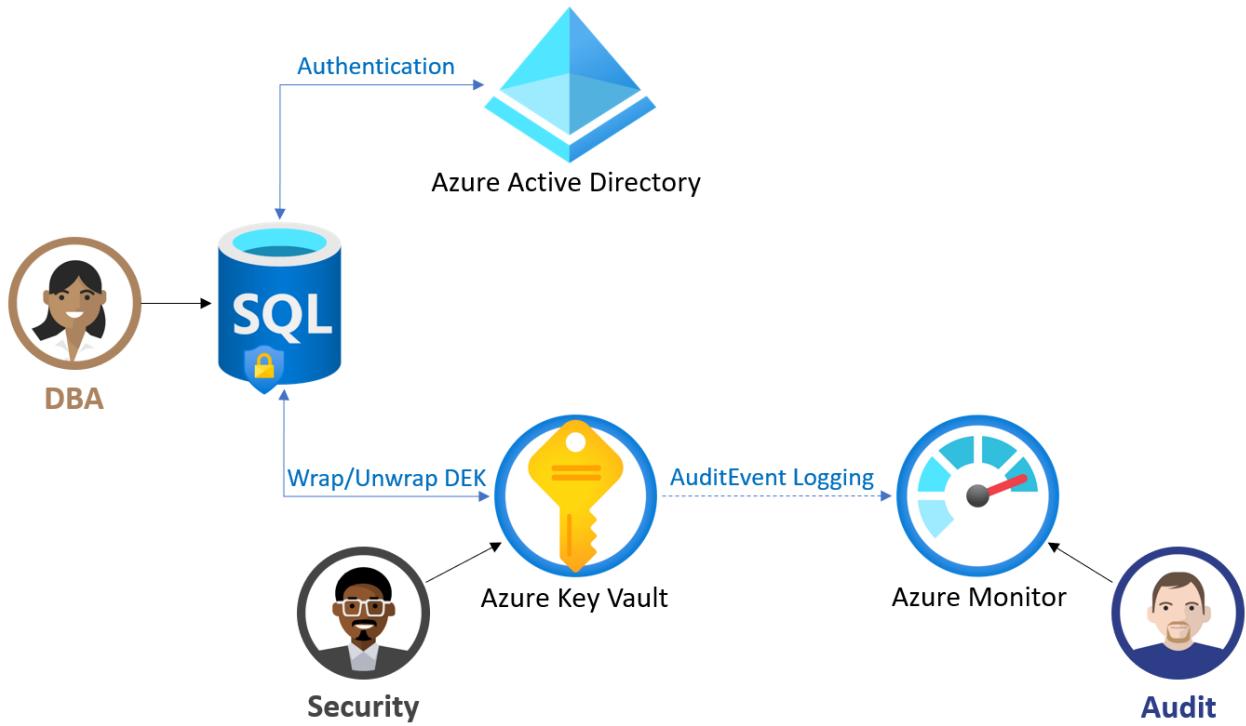
For those using service-managed TDE who would like to start using customer-managed TDE, data remains encrypted during the process of switching over, and there is no downtime nor re-encryption of the database files. Switching from a service-managed key to a customer-managed key only requires re-encryption of the DEK, which is a fast and online operation.

Benefits of the customer-managed TDE

Customer-managed TDE provides the following benefits to the customer:

- Full and granular control over usage and management of the TDE protector;
- Transparency of the TDE protector usage;
- Ability to implement separation of duties in the management of keys and data within the organization;
- Key Vault administrator can revoke key access permissions to make encrypted database inaccessible;
- Central management of keys in AKV;
- Greater trust from your end customers, since AKV is designed such that Microsoft cannot see nor extract encryption keys;

How customer-managed TDE works



For server to be able to use TDE protector stored in AKV for encryption of the DEK, key vault administrator needs to give the following access rights to the server using its unique AAD identity:

- **get** - for retrieving the public part and properties of the key in the Key Vault
- **wrapKey** - to be able to protect (encrypt) DEK
- **unwrapKey** - to be able to unprotect (decrypt) DEK

Key vault administrator can also [enable logging of key vault audit events](#), so they can be audited later.

When server is configured to use a TDE protector from AKV, the server sends the DEK of each TDE-enabled database to the key vault for encryption. Key vault returns the encrypted DEK, which is then stored in the user database.

When needed, server sends protected DEK to the key vault for decryption.

Auditors can use Azure Monitor to review key vault AuditEvent logs, if logging is enabled.

NOTE

It may take around 10 minutes for any permission changes to take effect for the key vault. This includes revoking access permissions to the TDE protector in AKV, and users within this time frame may still have access permissions.

Requirements for configuring customer-managed TDE

Requirements for configuring AKV

- Key vault and SQL Database/managed instance must belong to the same Azure Active Directory tenant. Cross-tenant key vault and server interactions are not supported. To move resources afterwards, TDE with AKV will have to be reconfigured. Learn more about [moving resources](#).
- **Soft-delete** feature must be enabled on the key vault, to protect from data loss accidental key (or key vault) deletion happens. Soft-deleted resources are retained for 90 days, unless recovered or purged by the customer in the meantime. The *recover* and *purge* actions have their own permissions associated in a key vault access policy. Soft-delete feature is off by default and can be enabled via [Powershell](#) or [CLI](#). It cannot be enabled via Azure portal.

- Grant the SQL Database server or managed instance access to the key vault (get, wrapKey, unwrapKey) using its Azure Active Directory identity. When using Azure portal, the Azure AD identity gets automatically created. When using PowerShell or CLI, the Azure AD identity must be explicitly created and completion should be verified. See [Configure TDE with BYOK](#) and [Configure TDE with BYOK for Managed Instance](#) for detailed step-by-step instructions when using PowerShell.
- When using firewall with AKV, you must enable option *Allow trusted Microsoft services to bypass the firewall*.

Requirements for configuring TDE protector

- TDE protector can be only asymmetric, RSA 2048 or RSA HSM 2048 key.
- The key activation date (if set) must be a date and time in the past. Expiration date (if set) must be a future date and time.
- The key must be in the *Enabled* state.
- If you are importing existing key into the key vault, make sure to provide it in the supported file formats (.pfx, .byok, or .backup).

Recommendations when configuring customer-managed TDE

Recommendations when configuring AKV

- Associate at most 500 General Purpose or 200 Business Critical databases in total with a key vault in a single subscription to ensure high availability when server accesses the TDE protector in the key vault. These figures are based on the experience and documented in the [key vault service limits](#). The intention here is to prevent issues after server failover, as it will trigger as many key operations against the vault as there are databases in that server.
- Set a resource lock on the key vault to control who can delete this critical resource and prevent accidental or unauthorized deletion. Learn more about [resource locks](#).
- Enable auditing and reporting on all encryption keys: Key vault provides logs that are easy to inject into other security information and event management tools. Operations Management Suite [Log Analytics](#) is one example of a service that is already integrated.
- Link each server with two key vaults that reside in different regions and hold the same key material, to ensure high availability of encrypted databases. Mark only the key from the key vault in the same region as a TDE protector. System will use

Recommendations when configuring TDE protector

- Keep a copy of the TDE protector on a secure place or escrow it to the escrow service.
- If the key is generated in the key vault, create a key backup before using the key in AKV for the first time. Backup can be restored to an Azure Key Vault only. Learn more about the [Backup-AzKeyVaultKey](#) command.
- Create a new backup whenever any changes are made to the key (e.g. key attributes, tags, ACLs).
- **Keep previous versions** of the key in the key vault when rotating keys, so older database backups can be restored. When the TDE protector is changed for a database, old backups of the database **are not updated** to use the latest TDE protector. At restore time, each backup needs the TDE protector it was encrypted with at creation time. Key rotations can be performed following the instructions at [Rotate the Transparent Data Encryption Protector Using PowerShell](#).
- Keep all previously used keys in AKV even after switching to service-managed keys. It ensures database backups can be restored with the TDE protectors stored in AKV. TDE protectors created with Azure Key

Vault have to be maintained until all remaining stored backups have been created with service-managed keys. Make recoverable backup copies of these keys using [Backup-AzKeyVaultKey](#).

- To remove a potentially compromised key during a security incident without the risk of data loss, follow the steps from the [Remove a potentially compromised key](#).

Inaccessible TDE protector

When transparent data encryption is configured to use a customer-managed key, continuous access to the TDE protector is required for the database to stay online. If the server loses access to the customer-managed TDE protector in AKV, in up to 10 minutes a database will start denying all connections with the corresponding error message and change its state to *Inaccessible*. The only action allowed on a database in the Inaccessible state is deleting it.

NOTE

If the database is inaccessible due to an intermittent networking outage, there is no action required and the databases will come back online automatically.

After access to the key is restored, taking database back online requires additional time and steps, which may vary based on the time elapsed without access to the key and the size of the data in the database:

- If key access is restored within 8 hours, the database will auto-heal within next hour.
- If key access is restored after more than 8 hours, auto-heal is not possible and bringing the database back requires additional steps on the portal and can take a significant amount of time depending on the size of the database. Once the database is back online, previously configured server-level settings such as [failover group](#) configuration, point-in-time-restore history, and tags **will be lost**. Therefore, it's recommended implementing a notification system that allows you to identify and address the underlying key access issues within 8 hours.

Accidental TDE protector access revocation

It may happen that someone with sufficient access rights to the key vault accidentally disables server access to the key by:

- revoking key vault's *get*, *wrapKey*, *unwrapKey* permissions from the server
- deleting the key
- deleting the key vault
- changing key vault's firewall rules
- deleting the managed identity of the server in Azure Active Directory

Learn more about [the common causes for database to become inaccessible](#).

Monitoring of the customer-managed TDE

To monitor database state and to enable alerting for loss of TDE protector access, configure the following Azure features:

- [Azure Resource Health](#). An inaccessible database that has lost access to the TDE protector will show as "Unavailable" after the first connection to the database has been denied.
- [Activity Log](#) when access to the TDE protector in the customer-managed key vault fails, entries are added to the activity log. Creating alerts for these events will enable you to reinstate access as soon as possible.
- [Action Groups](#) can be defined to send you notifications and alerts based on your preferences, e.g.

Email/SMS/Push/Voice, Logic App, Webhook, ITSM, or Automation Runbook.

Database backup and restore with customer-managed TDE

Once a database is encrypted with TDE using a key from Key Vault, any newly generated backups are also encrypted with the same TDE protector. When the TDE protector is changed, old backups of the database **are not updated** to use the latest TDE protector.

To restore a backup encrypted with a TDE protector from Key Vault, make sure that the key material is available to the target server. Therefore, we recommend that you keep all the old versions of the TDE protector in key vault, so database backups can be restored.

IMPORTANT

At any moment there can be not more than one TDE protector set for a server. It's the key marked with "Make the key the default TDE protector" in the Azure portal blade. However, multiple additional keys can be linked to a server without marking them as a TDE protector. These keys are not used for protecting DEK, but can be used during restore from a backup, if backup file is encrypted with the key with the corresponding thumbprint.

If the key that is needed for restoring a backup is no longer available to the target server, the following error message is returned on the restore try: "Target server <Servername> does not have access to all AKV URIs created between <Timestamp #1> and <Timestamp #2>. Please retry operation after restoring all AKV URIs."

To mitigate it, run the [Get-AzSqlServerKeyVaultKey](#) cmdlet for target SQL Database logical server or [Get-AzSqlInstanceKeyVaultKey](#) for target managed instance to return the list of available keys and identify the missing ones. To ensure all backups can be restored, make sure the target server for the restore has access to all of keys needed. These keys don't need to be marked as TDE protector.

To learn more about backup recovery for SQL Database, see [Recover an Azure SQL database](#). To learn more about backup recovery for SQL Data Warehouse, see [Recover an Azure SQL Data Warehouse](#). For SQL Server's native backup/restore with managed instance, see [Quickstart: Restore a database to a Managed Instance](#)

Additional consideration for log files: Backed up log files remain encrypted with the original TDE protector, even if it was rotated and the database is now using a new TDE protector. At restore time, both keys will be needed to restore the database. If the log file is using a TDE protector stored in Azure Key Vault, this key will be needed at restore time, even if the database has been changed to use service-managed TDE in the meantime.

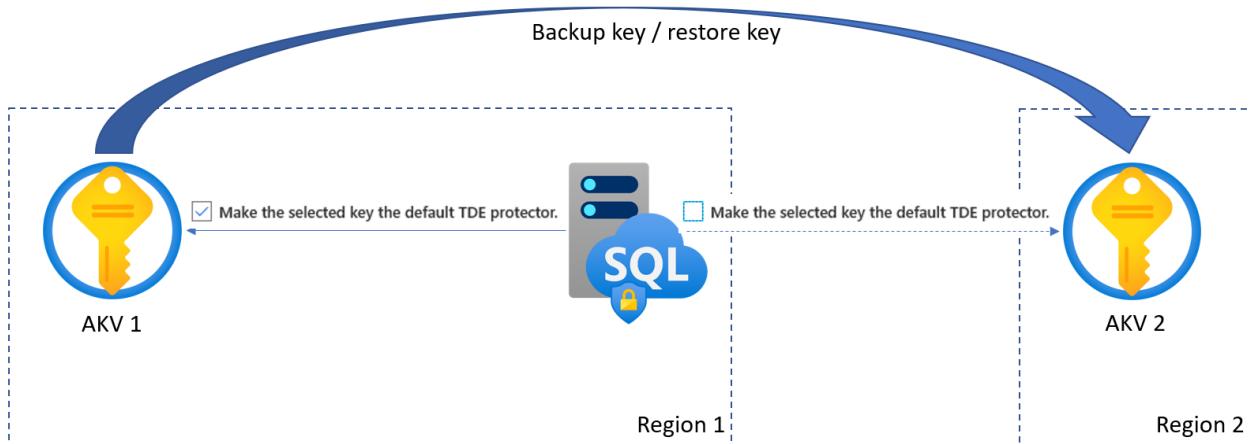
High availability with customer-managed TDE

Even in cases when there is no configured geo-redundancy for server, it is highly recommended to configure the server to use two different key vaults in two different regions with the same key material. It can be accomplished by creating a TDE protector using the primary key vault co-located in the same region as the server and cloning the key into a key vault in a different Azure region, so that the server has access to a second key vault should the primary key vault experience an outage while the database is up and running.

Use the `Backup-AzKeyVaultKey` cmdlet to retrieve the key in encrypted format from the primary key vault and then use the `Restore-AzKeyVaultKey` cmdlet and specify a key vault in the second region to clone the key.

Alternatively, use Azure portal to backup and restore key. The key in the secondary key vault in se other region should not be marked as TDE protector, and it's not even allowed.

If there is an outage affecting primary key vault, and only then, system will automatically switch to the other linked key with the same thumbprint in the secondary key vault, if it exists. Note though that switch will not happen if TDE protector is inaccessible because of revoked access rights, or because key or key vault is deleted, as it may indicate that customer intentionally wanted to restrict server from accessing the key.

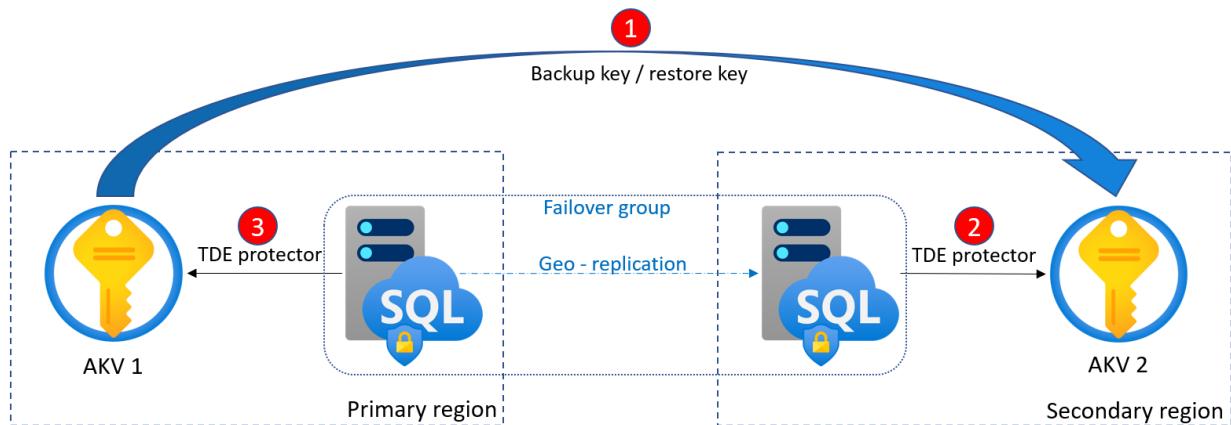


Geo-DR and customer-managed TDE

In both [active geo-replication](#) and [failover groups](#) scenarios, each server involved requires a separate key vault, that must be co-located with the server in the same Azure region. Customer is responsible for keeping the key material across the key vaults consistent, so that geo-secondary is in sync and can take over using the same key from its local key vault if primary becomes inaccessible due to an outage in the region and a failover is triggered. Up to four secondaries can be configured, and chaining (secondaries of secondaries) is not supported.

To avoid issues while establishing or during geo-replication due to incomplete key material, it's important to follow these rules when configuring customer-managed TDE:

- All key vaults involved must have same properties, and same access rights for respective servers.
- All key vaults involved must contain identical key material. It applies not just to the current TDE protector, but to all previous TDE protectors that may be used in the backup files.
- Both initial setup and rotation of the TDE protector must be done on the secondary first, and then on primary.



To test a failover, follow the steps in [Active geo-replication overview](#). It should be done on a regular basis to confirm the access permissions for SQL to both key vaults have been maintained.

Next steps

You may also want to check the following PowerShell sample scripts for the common operations with customer-managed TDE:

- [Rotate the Transparent Data Encryption Protector for SQL Database Using PowerShell](#)
- [Remove a Transparent Data Encryption \(TDE\) protector for SQL Database using PowerShell](#)

- Manage Transparent Data Encryption in a Managed Instance with your own key using PowerShell

PowerShell and CLI: Enable Transparent Data Encryption with customer-managed key from Azure Key Vault

1/23/2020 • 5 minutes to read • [Edit Online](#)

This article walks through how to use a key from Azure Key Vault for Transparent Data Encryption (TDE) on a SQL Database or Data Warehouse. To learn more about the TDE with Azure Key Vault integration - Bring Your Own Key (BYOK) Support, visit [TDE with customer-managed keys in Azure Key Vault](#).

Prerequisites for PowerShell

- You must have an Azure subscription and be an administrator on that subscription.
- [Recommended but Optional] Have a hardware security module (HSM) or local key store for creating a local copy of the TDE Protector key material.
- You must have Azure PowerShell installed and running.
- Create an Azure Key Vault and Key to use for TDE.
 - [Instructions for using a hardware security module \(HSM\) and Key Vault](#)
 - The key vault must have the following property to be used for TDE:
 - [soft-delete](#) and purge protection
- The key must have the following attributes to be used for TDE:
 - No expiration date
 - Not disabled
 - Able to perform *get*, *wrap key*, *unwrap key* operations
- [PowerShell](#)
- [Azure CLI](#)

For Az module installation instructions, see [Install Azure PowerShell](#). For specific cmdlets, see [AzureRM.Sql](#).

For specifics on Key Vault, see [PowerShell instructions from Key Vault](#) and [How to use Key Vault soft-delete with PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

Assign an Azure AD identity to your server

If you have an existing server, use the following to add an Azure AD identity to your server:

```
$server = Set-AzSqlServer -ResourceGroupName <SQLDatabaseResourceGroupName> -ServerName <LogicalServerName> -  
AssignIdentity
```

If you are creating a server, use the [New-AzSqlServer](#) cmdlet with the tag `-Identity` to add an Azure AD identity

during server creation:

```
$server = New-AzSqlServer -ResourceGroupName <SQLDatabaseResourceGroupName> -Location <RegionName> ` -ServerName <LogicalServerName> -ServerVersion "12.0" -SqlAdministratorCredentials <PSCredential> -AssignIdentity
```

Grant Key Vault permissions to your server

Use the [Set-AzKeyVaultAccessPolicy](#) cmdlet to grant your server access to the key vault before using a key from it for TDE.

```
Set-AzKeyVaultAccessPolicy -VaultName <KeyVaultName> ` -ObjectId $server.Identity.PrincipalId -PermissionsToKeys get, wrapKey, unwrapKey
```

Add the Key Vault key to the server and set the TDE Protector

- Use the [Get-AzKeyVaultKey](#) cmdlet to retrieve the key ID from key vault
- Use the [Add-AzSqlServerKeyVaultKey](#) cmdlet to add the key from the Key Vault to the server.
- Use the [Set-AzSqlServerTransparentDataEncryptionProtector](#) cmdlet to set the key as the TDE protector for all server resources.
- Use the [Get-AzSqlServerTransparentDataEncryptionProtector](#) cmdlet to confirm that the TDE protector was configured as intended.

NOTE

The combined length for the key vault name and key name cannot exceed 94 characters.

TIP

An example KeyId from Key Vault: <https://contosokeyvault.vault.azure.net/keys/Key1/1a1a2b2b3c3c4d4d5e5e6f6f7g7g8h8h>

```
# add the key from Key Vault to the server  
Add-AzSqlServerKeyVaultKey -ResourceGroupName <SQLDatabaseResourceGroupName> -ServerName <LogicalServerName> -KeyId <KeyVaultKeyId>  
  
# set the key as the TDE protector for all resources under the server  
Set-AzSqlServerTransparentDataEncryptionProtector -ResourceGroupName <SQLDatabaseResourceGroupName> -ServerName <LogicalServerName> ` -Type AzureKeyVault -KeyId <KeyVaultKeyId>  
  
# confirm the TDE protector was configured as intended  
Get-AzSqlServerTransparentDataEncryptionProtector -ResourceGroupName <SQLDatabaseResourceGroupName> -ServerName <LogicalServerName>
```

Turn on TDE

Use the [Set-AzSqlDatabaseTransparentDataEncryption](#) cmdlet to turn on TDE.

```
Set-AzSqlDatabaseTransparentDataEncryption -ResourceGroupName <SQLDatabaseResourceGroupName> ` -ServerName <LogicalServerName> -DatabaseName <DatabaseName> -State "Enabled"
```

Now the database or data warehouse has TDE enabled with an encryption key in Key Vault.

Check the encryption state and encryption activity

Use the [Get-AzSqlDatabaseTransparentDataEncryption](#) to get the encryption state and the [Get-AzSqlDatabaseTransparentDataEncryptionActivity](#) to check the encryption progress for a database or data warehouse.

```
# get the encryption state  
Get-AzSqlDatabaseTransparentDataEncryption -ResourceGroupName <SQLDatabaseResourceGroupName> `  
    -ServerName <LogicalServerName> -DatabaseName <DatabaseName> `  
  
# check the encryption progress for a database or data warehouse  
Get-AzSqlDatabaseTransparentDataEncryptionActivity -ResourceGroupName <SQLDatabaseResourceGroupName> `  
    -ServerName <LogicalServerName> -DatabaseName <DatabaseName>
```

Useful PowerShell cmdlets

- [PowerShell](#)
- [Azure CLI](#)
- Use the [Set-AzSqlDatabaseTransparentDataEncryption](#) cmdlet to turn off TDE.

```
Set-AzSqlDatabaseTransparentDataEncryption -ServerName <LogicalServerName> -ResourceGroupName  
<SQLDatabaseResourceGroupName> `  
    -DatabaseName <DatabaseName> -State "Disabled"
```

- Use the [Get-AzSqlServerKeyVaultKey](#) cmdlet to return the list of Key Vault keys added to the server.

```
# KeyId is an optional parameter, to return a specific key version  
Get-AzSqlServerKeyVaultKey -ServerName <LogicalServerName> -ResourceGroupName  
<SQLDatabaseResourceGroupName>
```

- Use the [Remove-AzSqlServerKeyVaultKey](#) to remove a Key Vault key from the server.

```
# the key set as the TDE Protector cannot be removed  
Remove-AzSqlServerKeyVaultKey -KeyId <KeyVaultKeyId> -ServerName <LogicalServerName> -ResourceGroupName  
<SQLDatabaseResourceGroupName>
```

Troubleshooting

Check the following if an issue occurs:

- If the key vault cannot be found, make sure you're in the right subscription.
 - [PowerShell](#)
 - [Azure CLI](#)

```
Get-AzSubscription -SubscriptionId <SubscriptionId>
```

- If the new key cannot be added to the server, or the new key cannot be updated as the TDE Protector, check the following:
 - The key should not have an expiration date

- The key must have the *get*, *wrap key*, and *unwrap key* operations enabled.

Next steps

- Learn how to rotate the TDE Protector of a server to comply with security requirements: [Rotate the Transparent Data Encryption protector Using PowerShell](#).
- In case of a security risk, learn how to remove a potentially compromised TDE Protector: [Remove a potentially compromised key](#).

Rotate the Transparent Data Encryption (TDE) protector using PowerShell

12/10/2019 • 3 minutes to read • [Edit Online](#)

This article describes key rotation for an Azure SQL server using a TDE protector from Azure Key Vault. Rotating an Azure SQL server's TDE protector means switching to a new asymmetric key that protects the databases on the server. Key rotation is an online operation and should only take a few seconds to complete, because this only decrypts and re-encrypts the database's data encryption key, not the entire database.

This guide discusses two options to rotate the TDE protector on the server.

NOTE

A paused SQL Data Warehouse must be resumed before key rotations.

IMPORTANT

Do not delete previous versions of the key after a rollover. When keys are rolled over, some data is still encrypted with the previous keys, such as older database backups.

Prerequisites

- This how-to guide assumes that you are already using a key from Azure Key Vault as the TDE protector for an Azure SQL Database or Data Warehouse. See [Transparent Data Encryption with BYOK Support](#).
- You must have Azure PowerShell installed and running.
- [Recommended but optional] Create the key material for the TDE protector in a hardware security module (HSM) or local key store first, and import the key material to Azure Key Vault. Follow the [instructions for using a hardware security module \(HSM\) and Key Vault](#) to learn more.
- [PowerShell](#)
- [Azure CLI](#)

For Az module installation instructions, see [Install Azure PowerShell](#). For specific cmdlets, see [AzureRM.Sql](#).

IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

Manual key rotation

Manual key rotation uses the following commands to add a completely new key, which could be under a new key name or even another key vault. Using this approach supports adding the same key to different key vaults to support high-availability and geo-dr scenarios.

NOTE

The combined length for the key vault name and key name cannot exceed 94 characters.

- [PowerShell](#)
- [Azure CLI](#)

Use the [Add-AzKeyVaultKey](#), [Add-AzSqlServerKeyVaultKey](#), and [Set-AzSqlServerTransparentDataEncryptionProtector](#) cmdlets.

```
# add a new key to Key Vault
Add-AzKeyVaultKey -VaultName <keyVaultName> -Name <keyVaultKeyName> -Destination <hardwareOrSoftware>

# add the new key from Key Vault to the server
Add-AzSqlServerKeyVaultKey -KeyId <keyVaultKeyId> -ServerName <logicalServerName> -ResourceGroup
<SQLDatabaseResourceGroupName>

# set the key as the TDE protector for all resources under the server
Set-AzSqlServerTransparentDataEncryptionProtector -Type AzureKeyVault -KeyId <keyVaultKeyId> ` 
    -ServerName <logicalServerName> -ResourceGroup <SQLDatabaseResourceGroupName>
```

Useful PowerShell cmdlets

- [PowerShell](#)
- [Azure CLI](#)
- To switch the TDE protector from Microsoft-managed to BYOK mode, use the [Set-AzSqlServerTransparentDataEncryptionProtector](#) cmdlet.

```
Set-AzSqlServerTransparentDataEncryptionProtector -Type AzureKeyVault ` 
    -KeyId <keyVaultKeyId> -ServerName <logicalServerName> -ResourceGroup
<SQLDatabaseResourceGroupName>
```

- To switch the TDE protector from BYOK mode to Microsoft-managed, use the [Set-AzSqlServerTransparentDataEncryptionProtector](#) cmdlet.

```
Set-AzSqlServerTransparentDataEncryptionProtector -Type ServiceManaged ` 
    -ServerName <logicalServerName> -ResourceGroup <SQLDatabaseResourceGroupName>
```

Next steps

- In case of a security risk, learn how to remove a potentially compromised TDE protector: [Remove a potentially compromised key](#)
- Get started with Azure Key Vault integration and Bring Your Own Key support for TDE: [Turn on TDE using your own key from Key Vault using PowerShell](#)

Remove a Transparent Data Encryption (TDE) protector using PowerShell

2/24/2020 • 5 minutes to read • [Edit Online](#)

Prerequisites

- You must have an Azure subscription and be an administrator on that subscription
- You must have Azure PowerShell installed and running.
- This how-to guide assumes that you are already using a key from Azure Key Vault as the TDE protector for an Azure SQL Database or Data Warehouse. See [Transparent Data Encryption with BYOK Support](#) to learn more.
- [PowerShell](#)
- [Azure CLI](#)

For Az module installation instructions, see [Install Azure PowerShell](#). For specific cmdlets, see [AzureRM.Sql](#).

IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

Overview

This how-to guide describes how to respond to a potentially compromised TDE protector for an Azure SQL Database or Data Warehouse that is using TDE with customer-managed keys in Azure Key Vault - Bring Your Own Key (BYOK) support. To learn more about BYOK support for TDE, see the [overview page](#).

The following procedures should only be done in extreme cases or in test environments. Review the how-to guide carefully, as deleting actively used TDE protectors from Azure Key Vault will result in **database unavailability**.

If a key is ever suspected to be compromised, such that a service or user had unauthorized access to the key, it's best to delete the key.

Keep in mind that once the TDE protector is deleted in Key Vault, in up to 10 minutes all encrypted databases will start denying all connections with the corresponding error message and change its state to [Inaccessible](#).

The following steps outline how to check the TDE Protector thumbprints still in use by Virtual Log Files (VLF) of a given database. The thumbprint of the current TDE protector of the database, and the database ID can be found by running:

```
SELECT [database_id],  
       [encryption_state],  
       [encryptor_type], /*asymmetric key means AKV, certificate means service-managed keys*/  
       [encryptor_thumbprint],  
FROM [sys].[dm_database_encryption_keys]
```

The following query returns the VLFs and the encryptor respective thumbprints in use. Each different thumbprint refers to different key in Azure Key Vault (AKV):

```
SELECT * FROM sys.dm_db_log_info (database_id)
```

- [PowerShell](#)
- [Azure CLI](#)

The PowerShell command **Get-AzureRmSqlServerKeyVaultKey** provides the thumbprint of the TDE Protector used in the query, so you can see which keys to keep and which keys to delete in AKV. Only keys no longer used by the database can be safely deleted from Azure Key Vault.

This how-to guide goes over two approaches depending on the desired result after the incident response:

- To keep the Azure SQL databases / Data Warehouses **accessible**
- To make the Azure SQL databases / Data Warehouses **inaccessible**

To keep the encrypted resources accessible

- [PowerShell](#)
- [Azure CLI](#)

1. Create a [new key in Key Vault](#). Make sure this new key is created in a separate key vault from the potentially compromised TDE protector, since access control is provisioned on a vault level.
2. Add the new key to the server using the [Add-AzSqlServerKeyVaultKey](#) and [Set-AzSqlServerTransparentDataEncryptionProtector](#) cmdlets and update it as the server's new TDE protector.

```
# add the key from Key Vault to the server
Add-AzSqlServerKeyVaultKey -ResourceGroupName <SQLDatabaseResourceGroupName> -ServerName
<LogicalServerName> -KeyId <KeyVaultKeyId>

# set the key as the TDE protector for all resources under the server
Set-AzSqlServerTransparentDataEncryptionProtector -ResourceGroupName <SQLDatabaseResourceGroupName>
-ServerName <LogicalServerName> -Type AzureKeyVault -KeyId <KeyVaultKeyId>
```

3. Make sure the server and any replicas have updated to the new TDE protector using the [Get-AzSqlServerTransparentDataEncryptionProtector](#) cmdlet.

NOTE

It may take a few minutes for the new TDE protector to propagate to all databases and secondary databases under the server.

```
Get-AzSqlServerTransparentDataEncryptionProtector -ServerName <LogicalServerName> -ResourceGroupName
<SQLDatabaseResourceGroupName>
```

4. Take a [backup of the new key](#) in Key Vault.

```
# -OutputFile parameter is optional; if removed, a file name is automatically generated.
Backup-AzKeyVaultKey -VaultName <KeyVaultName> -Name <KeyVaultKeyName> -OutputFile
<DesiredBackupFilePath>
```

5. Delete the compromised key from Key Vault using the [Remove-AzKeyVaultKey](#) cmdlet.

```
Remove-AzKeyVaultKey -VaultName <KeyVaultName> -Name <KeyVaultKeyName>
```

6. To restore a key to Key Vault in the future using the [Restore-AzKeyVaultKey](#) cmdlet:

```
Restore-AzKeyVaultKey -VaultName <KeyVaultName> -InputFile <BackupFilePath>
```

To make the encrypted resources inaccessible

1. Drop the databases that are being encrypted by the potentially compromised key.

The database and log files are automatically backed up, so a point-in-time restore of the database can be done at any point (as long as you provide the key). The databases must be dropped before deletion of an active TDE protector to prevent potential data loss of up to 10 minutes of the most recent transactions.

2. Back up the key material of the TDE protector in Key Vault.
3. Remove the potentially compromised key from Key Vault

NOTE

It may take around 10 minutes for any permission changes to take effect for the key vault. This includes revoking access permissions to the TDE protector in AKV, and users within this time frame may still have access permissions.

Next steps

- Learn how to rotate the TDE protector of a server to comply with security requirements: [Rotate the Transparent Data Encryption protector Using PowerShell](#)
- Get started with Bring Your Own Key support for TDE: [Turn on TDE using your own key from Key Vault using PowerShell](#)

Dynamic data masking for Azure SQL Database and Data Warehouse

1/23/2020 • 3 minutes to read • [Edit Online](#)

SQL Database dynamic data masking limits sensitive data exposure by masking it to non-privileged users.

Dynamic data masking helps prevent unauthorized access to sensitive data by enabling customers to designate how much of the sensitive data to reveal with minimal impact on the application layer. It's a policy-based security feature that hides the sensitive data in the result set of a query over designated database fields, while the data in the database is not changed.

For example, a service representative at a call center may identify callers by several digits of their credit card number, but those data items should not be fully exposed to the service representative. A masking rule can be defined that masks all but the last four digits of any credit card number in the result set of any query. As another example, an appropriate data mask can be defined to protect personally identifiable information (PII) data, so that a developer can query production environments for troubleshooting purposes without violating compliance regulations.

Dynamic data masking basics

You set up a dynamic data masking policy in the Azure portal by selecting the dynamic data masking operation in your SQL Database configuration blade or settings blade. This feature cannot be set by using portal for SQL DW (Please use Powershell or REST API)

Dynamic data masking permissions

Dynamic data masking can be configured by the Azure SQL Database admin, server admin, or [SQL Security Manager](#) roles.

Dynamic data masking policy

- **SQL users excluded from masking** - A set of SQL users or AAD identities that get unmasked data in the SQL query results. Users with administrator privileges are always excluded from masking, and see the original data without any mask.
- **Masking rules** - A set of rules that define the designated fields to be masked and the masking function that is used. The designated fields can be defined using a database schema name, table name, and column name.
- **Masking functions** - A set of methods that control the exposure of data for different scenarios.

MASKING FUNCTION

MASKING LOGIC

MASKING FUNCTION	MASKING LOGIC						
Default	<p>Full masking according to the data types of the designated fields</p> <ul style="list-style-type: none"> • Use XXXX or fewer Xs if the size of the field is less than 4 characters for string data types (nchar, ntext, nvarchar). • Use a zero value for numeric data types (bigint, bit, decimal, int, money, numeric, smallint, smallmoney, tinyint, float, real). • Use 01-01-1900 for date/time data types (date, datetime2, datetime, datetimeoffset, smalldatetime, time). • For SQL variant, the default value of the current type is used. • For XML the document <masked/> is used. • Use an empty value for special data types (timestamp table, hierarchyid, GUID, binary, image, varbinary spatial types). 						
Credit card	<p>Masking method, which exposes the last four digits of the designated fields and adds a constant string as a prefix in the form of a credit card.</p> <p>XXXX-XXXX-XXXX-1234</p>						
Email	<p>Masking method, which exposes the first letter and replaces the domain with XXX.com using a constant string prefix in the form of an email address.</p> <p>aXX@XXXX.com</p>						
Random number	<p>Masking method, which generates a random number according to the selected boundaries and actual data types. If the designated boundaries are equal, then the masking function is a constant number.</p> <p>Masking Field Format <input type="button" value="Random number"/>▼</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50px; padding: 2px;">From</td> <td style="width: 50px; padding: 2px;">To</td> </tr> <tr> <td style="padding: 2px;"><input type="text" value="0"/> ✓</td> <td style="padding: 2px;"><input type="text" value="0"/> ✓</td> </tr> </table>	From	To	<input type="text" value="0"/> ✓	<input type="text" value="0"/> ✓		
From	To						
<input type="text" value="0"/> ✓	<input type="text" value="0"/> ✓						
Custom text	<p>Masking method, which exposes the first and last characters and adds a custom padding string in the middle. If the original string is shorter than the exposed prefix and suffix, only the padding string is used. prefix[padding]suffix</p> <p>Masking Field Format <input type="button" value="Custom text"/>▼</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33.33%; padding: 2px;">Exposed Prefix</td> <td style="width: 33.33%; padding: 2px;">Padding String</td> <td style="width: 33.33%; padding: 2px;">Exposed Suffix</td> </tr> <tr> <td style="padding: 2px;"><input type="text" value="3"/> ✓</td> <td style="padding: 2px;"><input type="text" value="X*X*X"/> ✓</td> <td style="padding: 2px;"><input type="text" value="2"/> ✓</td> </tr> </table>	Exposed Prefix	Padding String	Exposed Suffix	<input type="text" value="3"/> ✓	<input type="text" value="X*X*X"/> ✓	<input type="text" value="2"/> ✓
Exposed Prefix	Padding String	Exposed Suffix					
<input type="text" value="3"/> ✓	<input type="text" value="X*X*X"/> ✓	<input type="text" value="2"/> ✓					

Recommended fields to mask

The DDM recommendations engine, flags certain fields from your database as potentially sensitive fields, which may be good candidates for masking. In the Dynamic Data Masking blade in the portal, you will see the recommended columns for your database. All you need to do is click **Add Mask** for one or more columns and then **Save** to apply a mask for these fields.

Set up dynamic data masking for your database using PowerShell cmdlets

See [Azure SQL Database Cmdlets](#).

Set up dynamic data masking for your database using REST API

See [Operations for Azure SQL Database](#).

Always Encrypted: Protect sensitive data and store encryption keys in the Windows certificate store

11/7/2019 • 12 minutes to read • [Edit Online](#)

This article shows you how to secure sensitive data in a SQL database with database encryption by using the [Always Encrypted Wizard](#) in [SQL Server Management Studio \(SSMS\)](#). It also shows you how to store your encryption keys in the Windows certificate store.

Always Encrypted is a new data encryption technology in Azure SQL Database and SQL Server that helps protect sensitive data at rest on the server, during movement between client and server, and while the data is in use, ensuring that sensitive data never appears as plaintext inside the database system. After you encrypt data, only client applications or app servers that have access to the keys can access plaintext data. For detailed information, see [Always Encrypted \(Database Engine\)](#).

After configuring the database to use Always Encrypted, you will create a client application in C# with Visual Studio to work with the encrypted data.

Follow the steps in this article to learn how to set up Always Encrypted for an Azure SQL database. In this article, you will learn how to perform the following tasks:

- Use the Always Encrypted wizard in SSMS to create [Always Encrypted Keys](#).
 - Create a [Column Master Key \(CMK\)](#).
 - Create a [Column Encryption Key \(CEK\)](#).
- Create a database table and encrypt columns.
- Create an application that inserts, selects, and displays data from the encrypted columns.

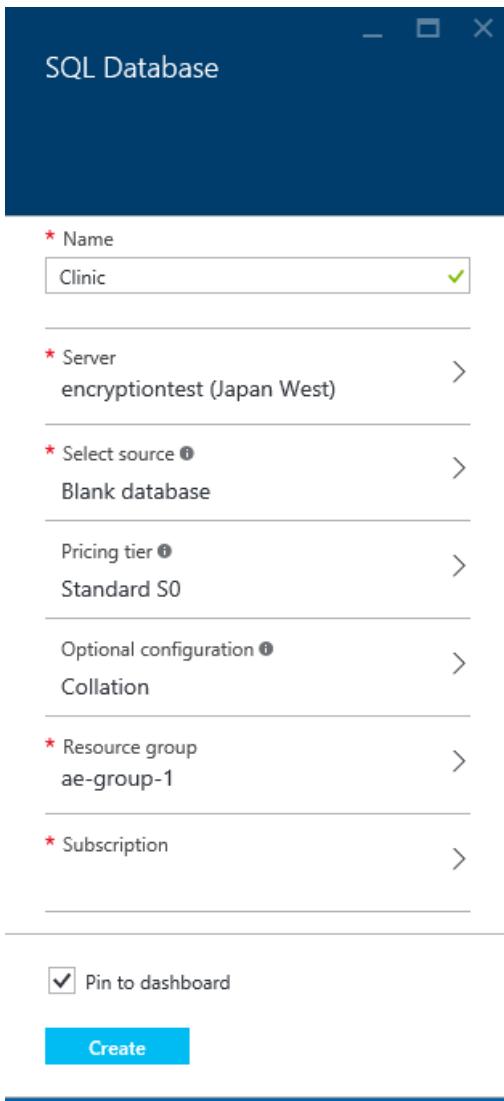
Prerequisites

For this tutorial, you'll need:

- An Azure account and subscription. If you don't have one, sign up for a [free trial](#).
- [SQL Server Management Studio](#) version 13.0.700.242 or later.
- [.NET Framework 4.6](#) or later (on the client computer).
- [Visual Studio](#).

Create a blank SQL database

1. Sign in to the [Azure portal](#).
2. Click **Create a resource > Data + Storage > SQL Database**.
3. Create a **Blank** database named **Clinic** on a new or existing server. For detailed instructions about creating a database in the Azure portal, see [Your first Azure SQL database](#).



You will need the connection string later in the tutorial. After the database is created, go to the new Clinic database and copy the connection string. You can get the connection string at any time, but it's easy to copy it when you're in the Azure portal.

1. Click **SQL databases** > **Clinic** > **Show database connection strings**.

2. Copy the connection string for **ADO.NET**.

The screenshot shows the 'Clinic' database details page in the Azure portal. The 'Show database connection strings' link is highlighted. The 'ADO.NET' section shows the connection string: Server=tcp:encryptiontest.database.windows.net,1433;Database=Clinic;User ID=sstein@encn. Other sections include ODBC (Includes Node.js), PHP, and JDBC.

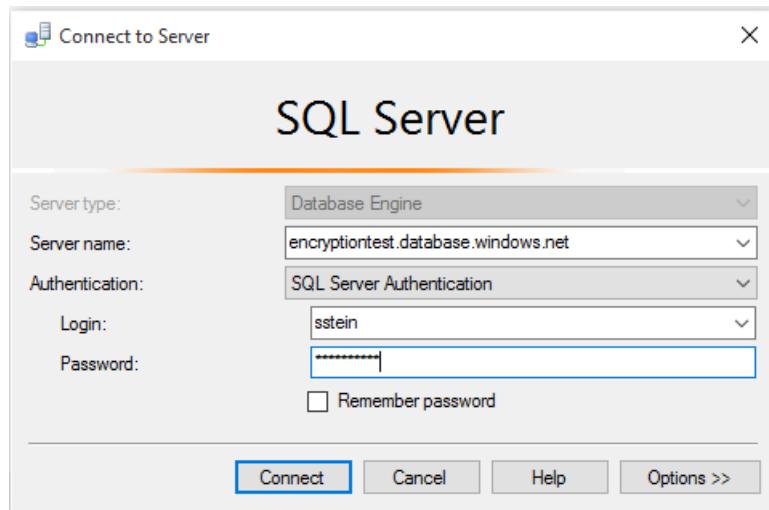
Connect to the database with SSMS

Open SSMS and connect to the server with the Clinic database.

1. Open SSMS. (Click **Connect** > **Database Engine** to open the **Connect to Server** window if it is not

open).

2. Enter your server name and credentials. The server name can be found on the SQL database blade and in the connection string you copied earlier. Type the complete server name including *database.windows.net*.



If the **New Firewall Rule** window opens, sign in to Azure and let SSMS create a new firewall rule for you.

Create a table

In this section, you will create a table to hold patient data. This will be a normal table initially--you will configure encryption in the next section.

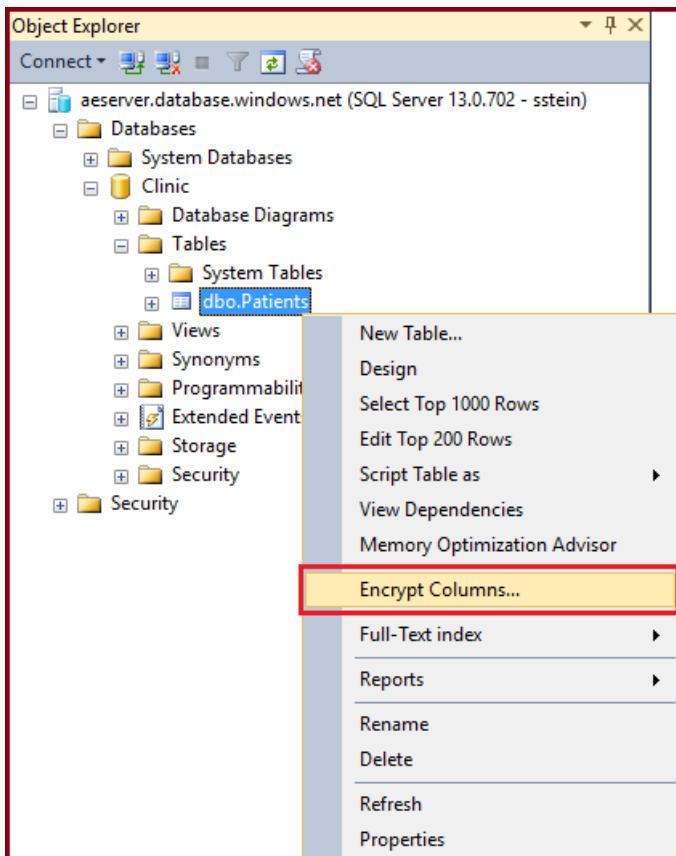
1. Expand **Databases**.
2. Right-click the **Clinic** database and click **New Query**.
3. Paste the following Transact-SQL (T-SQL) into the new query window and **Execute** it.

```
CREATE TABLE [dbo].[Patients]{
    [PatientId] [int] IDENTITY(1,1),
    [SSN] [char](11) NOT NULL,
    [FirstName] [nvarchar](50) NULL,
    [LastName] [nvarchar](50) NULL,
    [MiddleName] [nvarchar](50) NULL,
    [StreetAddress] [nvarchar](50) NULL,
    [City] [nvarchar](50) NULL,
    [ZipCode] [char](5) NULL,
    [State] [char](2) NULL,
    [BirthDate] [date] NOT NULL
    PRIMARY KEY CLUSTERED ([PatientId] ASC) ON [PRIMARY] );
GO
```

Encrypt columns (configure Always Encrypted)

SSMS provides a wizard to easily configure Always Encrypted by setting up the CMK, CEK, and encrypted columns for you.

1. Expand **Databases > Clinic > Tables**.
2. Right-click the **Patients** table and select **Encrypt Columns** to open the Always Encrypted wizard:



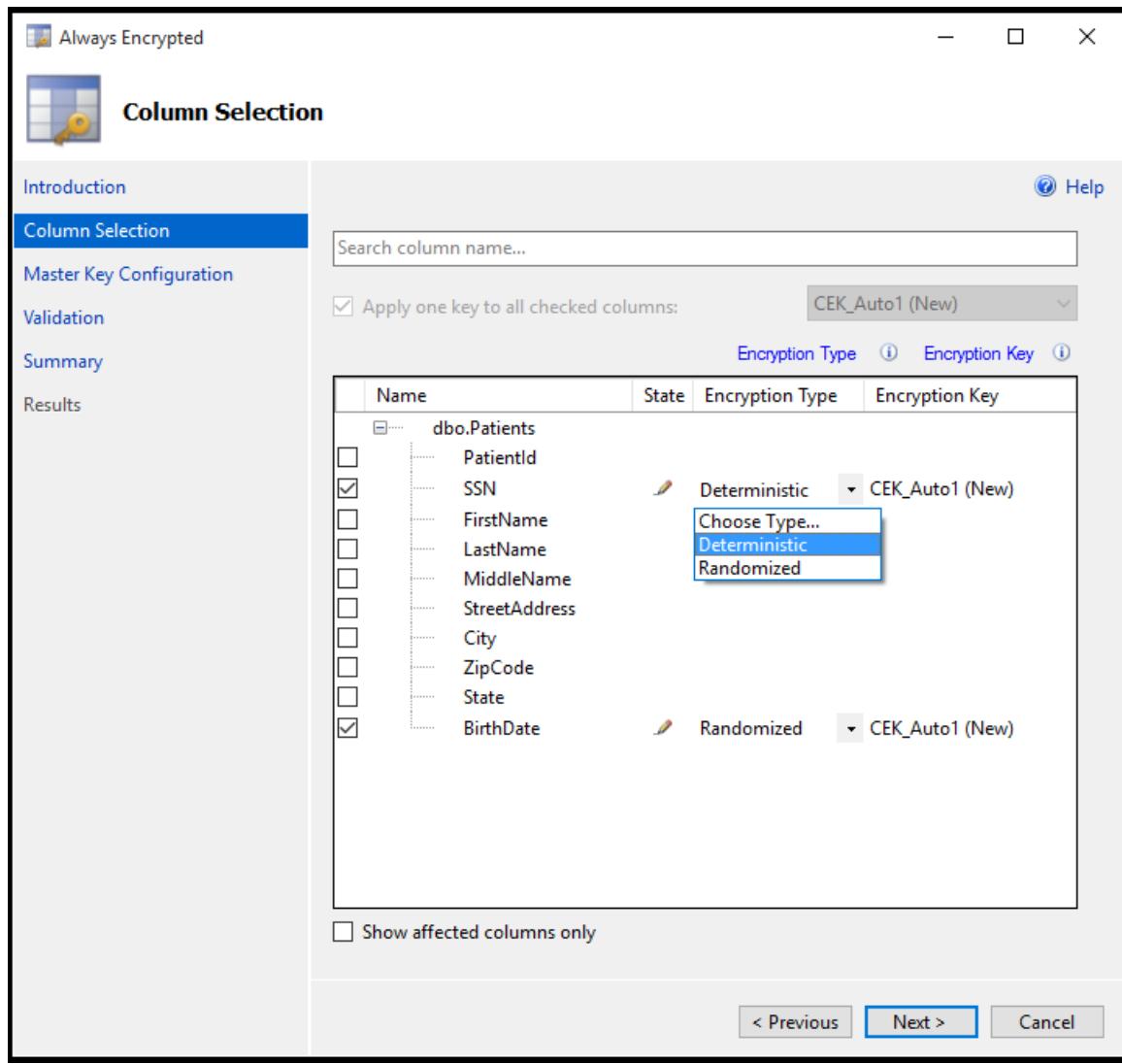
The Always Encrypted wizard includes the following sections: **Column Selection**, **Master Key Configuration (CMK)**, **Validation**, and **Summary**.

Column Selection

Click **Next** on the **Introduction** page to open the **Column Selection** page. On this page, you will select which columns you want to encrypt, [the type of encryption](#), and [what column encryption key \(CEK\)](#) to use.

Encrypt **SSN** and **BirthDate** information for each patient. The **SSN** column will use deterministic encryption, which supports equality lookups, joins, and group by. The **BirthDate** column will use randomized encryption, which does not support operations.

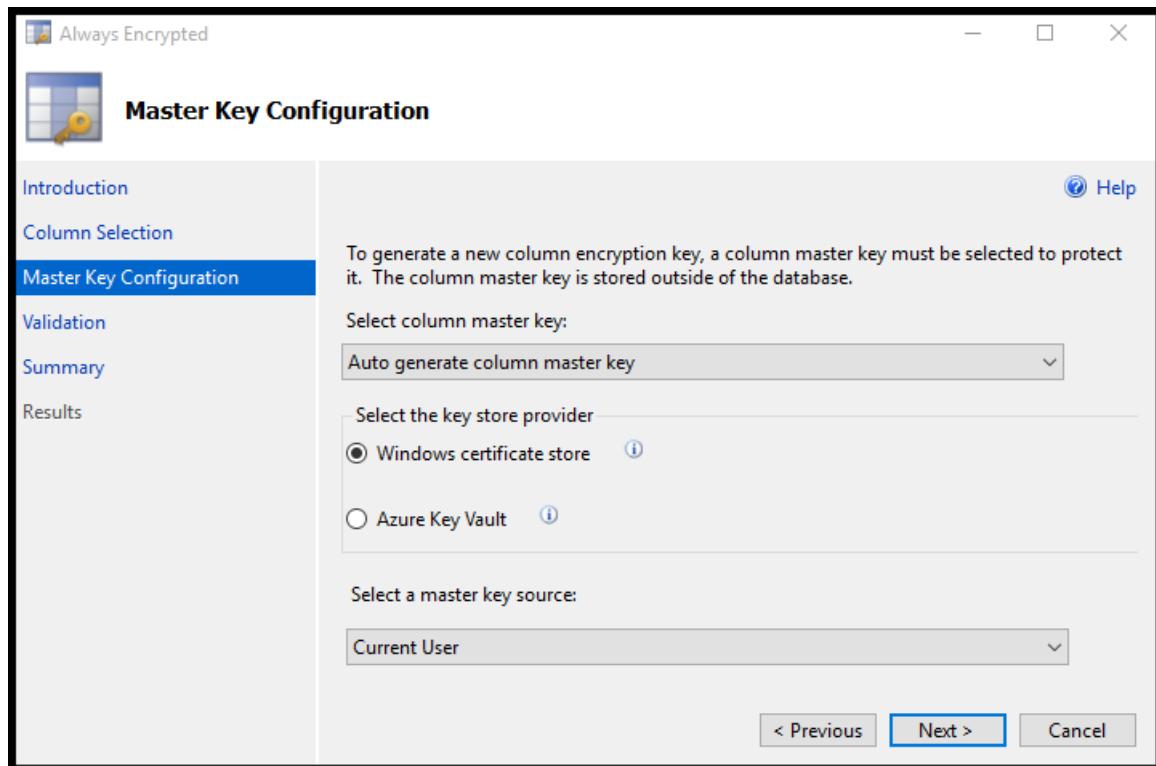
Set the **Encryption Type** for the **SSN** column to **Deterministic** and the **BirthDate** column to **Randomized**. Click **Next**.



Master Key Configuration

The **Master Key Configuration** page is where you set up your CMK and select the key store provider where the CMK will be stored. Currently, you can store a CMK in the Windows certificate store, Azure Key Vault, or a hardware security module (HSM). This tutorial shows how to store your keys in the Windows certificate store.

Verify that **Windows certificate store** is selected and click **Next**.

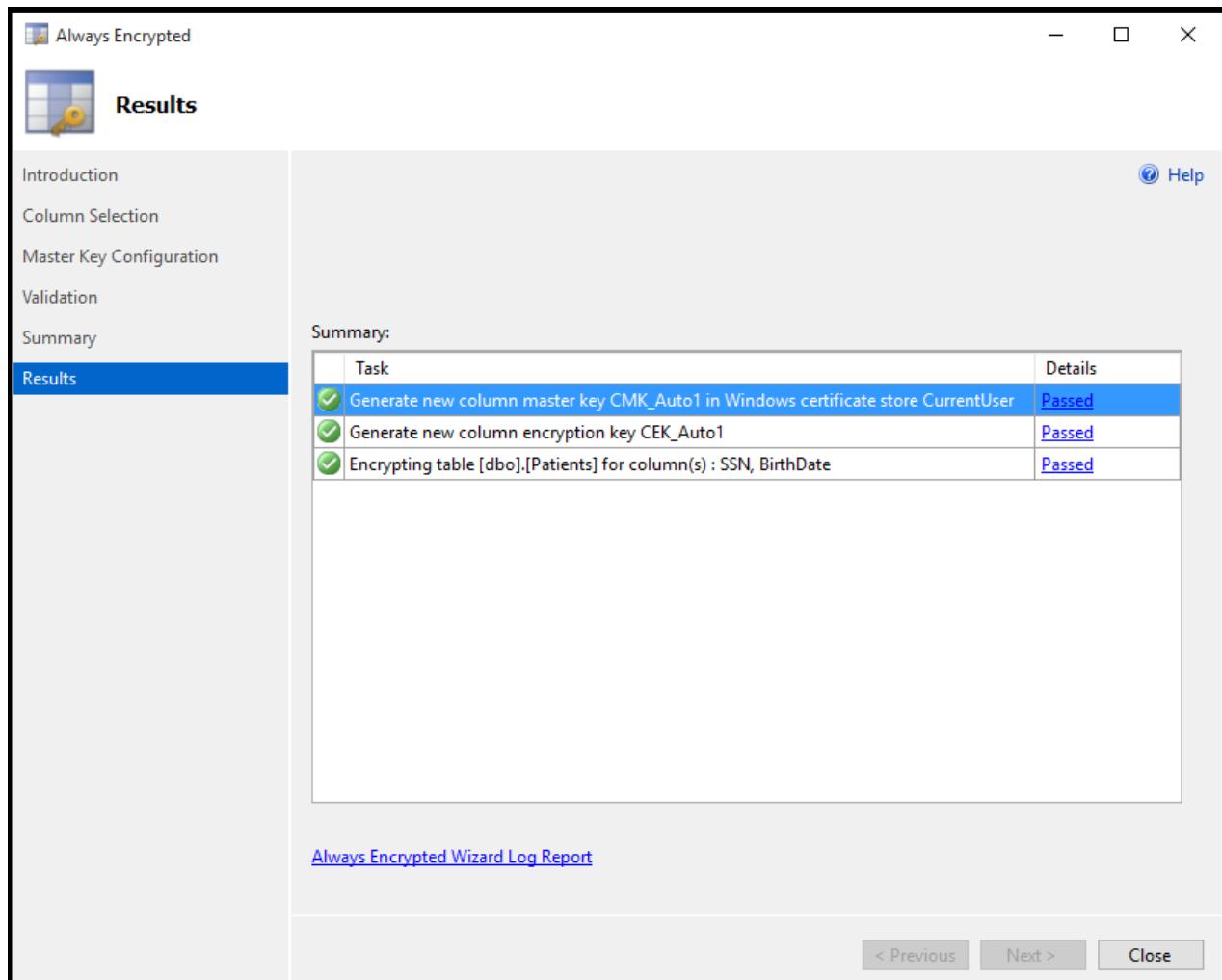


Validation

You can encrypt the columns now or save a PowerShell script to run later. For this tutorial, select **Proceed to finish now** and click **Next**.

Summary

Verify that the settings are all correct and click **Finish** to complete the setup for Always Encrypted.



Verify the wizard's actions

After the wizard is finished, your database is set up for Always Encrypted. The wizard performed the following actions:

- Created a CMK.
- Created a CEK.
- Configured the selected columns for encryption. Your **Patients** table currently has no data, but any existing data in the selected columns is now encrypted.

You can verify the creation of the keys in SSMS by going to **Clinic > Security > Always Encrypted Keys**. You can now see the new keys that the wizard generated for you.

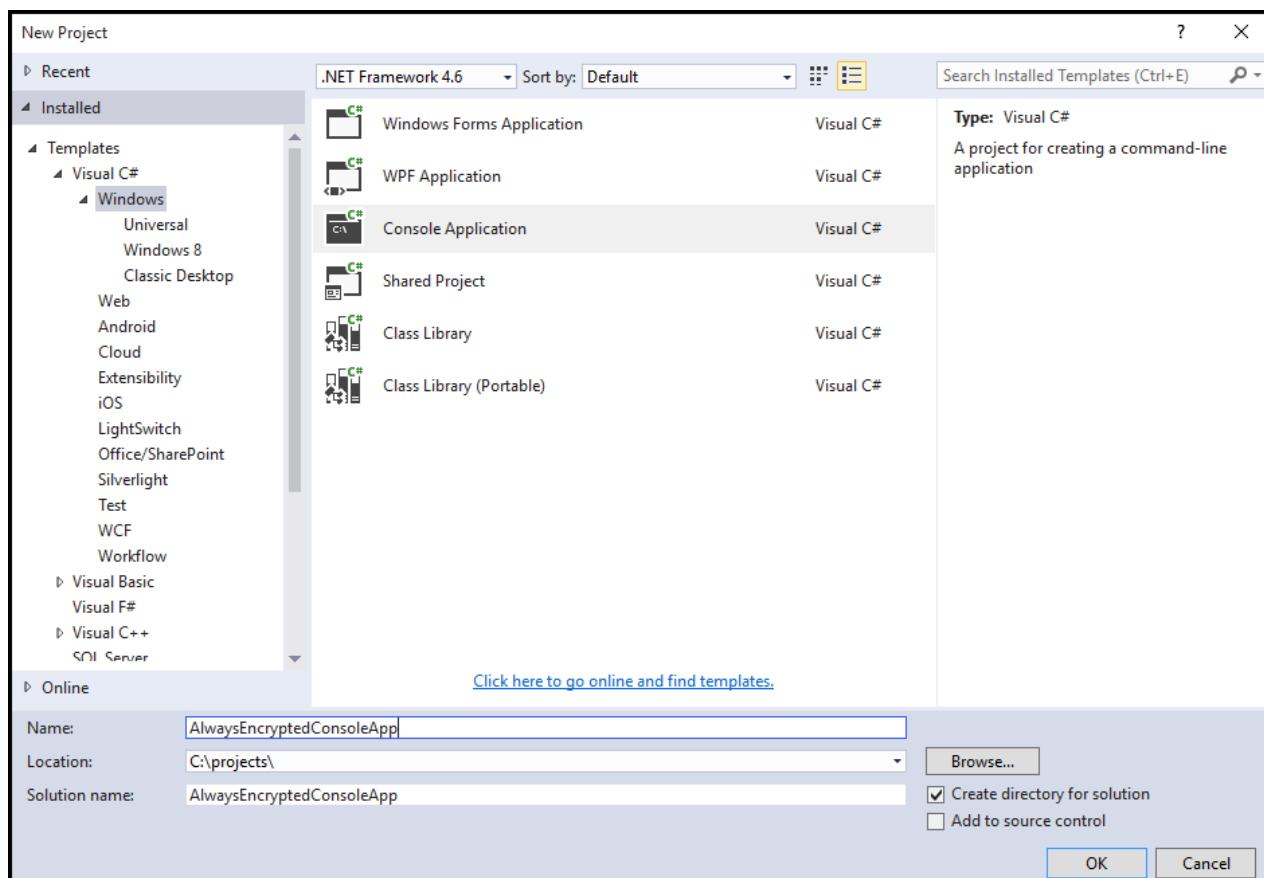
Create a client application that works with the encrypted data

Now that Always Encrypted is set up, you can build an application that performs *inserts* and *selects* on the encrypted columns. To successfully run the sample application, you must run it on the same computer where you ran the Always Encrypted wizard. To run the application on another computer, you must deploy your Always Encrypted certificates to the computer running the client app.

IMPORTANT

Your application must use `SqlParameter` objects when passing plaintext data to the server with Always Encrypted columns. Passing literal values without using `SqlParameter` objects will result in an exception.

1. Open Visual Studio and create a new C# console application. Make sure your project is set to **.NET Framework 4.6** or later.
2. Name the project **AlwaysEncryptedConsoleApp** and click **OK**.



Modify your connection string to enable Always Encrypted

This section explains how to enable Always Encrypted in your database connection string. You will modify the console app you just created in the next section, "Always Encrypted sample console application."

To enable Always Encrypted, you need to add the **Column Encryption Setting** keyword to your connection string and set it to **Enabled**.

You can set this directly in the connection string, or you can set it by using a [SqlConnectionStringBuilder](#). The sample application in the next section shows how to use [SqlConnectionStringBuilder](#).

NOTE

This is the only change required in a client application specific to Always Encrypted. If you have an existing application that stores its connection string externally (that is, in a config file), you might be able to enable Always Encrypted without changing any code.

Enable Always Encrypted in the connection string

Add the following keyword to your connection string:

```
Column Encryption Setting=Enabled
```

Enable Always Encrypted with a SqlConnectionStringBuilder

The following code shows how to enable Always Encrypted by setting the [SqlConnectionStringBuilder.ColumnEncryptionSetting](#) to [Enabled](#).

```
// Instantiate a SqlConnectionStringBuilder.  
SqlConnectionStringBuilder connStringBuilder =  
    new SqlConnectionStringBuilder("replace with your connection string");  
  
// Enable Always Encrypted.  
connStringBuilder.ColumnEncryptionSetting =  
    SqlConnectionColumnEncryptionSetting.Enabled;
```

Always Encrypted sample console application

This sample demonstrates how to:

- Modify your connection string to enable Always Encrypted.
- Insert data into the encrypted columns.
- Select a record by filtering for a specific value in an encrypted column.

Replace the contents of **Program.cs** with the following code. Replace the connection string for the global `connectionString` variable in the line directly above the `Main` method with your valid connection string from the Azure portal. This is the only change you need to make to this code.

Run the app to see Always Encrypted in action.

```
using System;  
using System.Collections.Generic;  
using System.Data;  
using System.Data.SqlClient;  
using System.Globalization;  
  
namespace AlwaysEncryptedConsoleApp  
{  
    class Program  
    {
```

```
// Update this line with your Clinic database connection string from the Azure portal.
static string connectionString = @"Data Source = SPE-T640-01.sys-sqlsvr.local; Initial Catalog =
Clinic; Integrated Security = true";

static void Main(string[] args)
{
    Console.WriteLine("Original connection string copied from the Azure portal:");
    Console.WriteLine(connectionString);

    // Create a SqlConnectionStringBuilder.
    SqlConnectionStringBuilder connStringBuilder =
        new SqlConnectionStringBuilder(connectionString);

    // Enable Always Encrypted for the connection.
    // This is the only change specific to Always Encrypted
    connStringBuilder.ColumnEncryptionSetting =
        SqlConnectionColumnEncryptionSetting.Enabled;

    Console.WriteLine(Environment.NewLine + "Updated connection string with Always Encrypted
enabled:");
    Console.WriteLine(connStringBuilder.ConnectionString);

    // Update the connection string with a password supplied at runtime.
    Console.WriteLine(Environment.NewLine + "Enter server password:");
    connStringBuilder.Password = Console.ReadLine();

    // Assign the updated connection string to our global variable.
    connectionString = connStringBuilder.ConnectionString;

    // Delete all records to restart this demo app.
    ResetPatientsTable();

    // Add sample data to the Patients table.
    Console.WriteLine(Environment.NewLine + "Adding sample patient data to the database...");

    CultureInfo culture = CultureInfo.CreateSpecificCulture("en-US");
    InsertPatient(new Patient()
    {
        SSN = "999-99-0001",
        FirstName = "Orlando",
        LastName = "Gee",
        BirthDate = DateTime.Parse("01/04/1964", culture)
    });
    InsertPatient(new Patient()
    {
        SSN = "999-99-0002",
        FirstName = "Keith",
        LastName = "Harris",
        BirthDate = DateTime.Parse("06/20/1977", culture)
    });
    InsertPatient(new Patient()
    {
        SSN = "999-99-0003",
        FirstName = "Donna",
        LastName = "Carreras",
        BirthDate = DateTime.Parse("02/09/1973", culture)
    });
    InsertPatient(new Patient()
    {
        SSN = "999-99-0004",
        FirstName = "Janet",
        LastName = "Gates",
        BirthDate = DateTime.Parse("08/31/1985", culture)
    });
    InsertPatient(new Patient()
    {
        SSN = "999-99-0005",
        FirstName = "Lucy",
        LastName = "Fitzgerald",
        BirthDate = DateTime.Parse("05/15/1988", culture)
    });
}
```

```

        LastName = "Harrington",
        BirthDate = DateTime.Parse("05/06/1993", culture)
    });

    // Fetch and display all patients.
    Console.WriteLine(Environment.NewLine + "All the records currently in the Patients table:");

    foreach (Patient patient in SelectAllPatients())
    {
        Console.WriteLine(patient.FirstName + " " + patient.LastName + "\tSSN: " + patient.SSN +
"\tBirthdate: " + patient.BirthDate);
    }

    // Get patients by SSN.
    Console.WriteLine(Environment.NewLine + "Now let's locate records by searching the encrypted SSN
column.");
}

string ssn;

// This very simple validation only checks that the user entered 11 characters.
// In production be sure to check all user input and use the best validation for your specific
application.
do
{
    Console.WriteLine("Please enter a valid SSN (ex. 123-45-6789):");
    ssn = Console.ReadLine();
} while (ssn.Length != 11);

// The example allows duplicate SSN entries so we will return all records
// that match the provided value and store the results in selectedPatients.
Patient selectedPatient = SelectPatientBySSN(ssn);

// Check if any records were returned and display our query results.
if (selectedPatient != null)
{
    Console.WriteLine("Patient found with SSN = " + ssn);
    Console.WriteLine(selectedPatient.FirstName + " " + selectedPatient.LastName + "\tSSN: "
+ selectedPatient.SSN + "\tBirthdate: " + selectedPatient.BirthDate);
}
else
{
    Console.WriteLine("No patients found with SSN = " + ssn);
}

Console.WriteLine("Press Enter to exit...");
Console.ReadLine();
}

static int InsertPatient(Patient newPatient)
{
    int returnValue = 0;

    string sqlCmdText = @"INSERT INTO [dbo].[Patients] ([SSN], [FirstName], [LastName], [BirthDate])
VALUES (@SSN, @FirstName, @LastName, @BirthDate);";

    SqlCommand sqlCmd = new SqlCommand(sqlCmdText);

    SqlParameter paramSSN = new SqlParameter("@@SSN", newPatient.SSN);
    paramSSN.DbType = DbType.AnsiStringFixedLength;
    paramSSN.Direction = ParameterDirection.Input;
    paramSSN.Size = 11;

    SqlParameter paramFirstName = new SqlParameter("@@FirstName", newPatient.FirstName);
    paramFirstName.DbType = DbType.String;
    paramFirstName.Direction = ParameterDirection.Input;
}

```

```

SqlParameter paramLastName = new SqlParameter("@@LastName", newPatient.LastName);
paramLastName.DbType = DbType.String;
paramLastName.Direction = ParameterDirection.Input;

SqlParameter paramBirthDate = new SqlParameter("@@BirthDate", newPatient.BirthDate);
paramBirthDate.SqlDbType = SqlDbType.Date;
paramBirthDate.Direction = ParameterDirection.Input;

sqlCmd.Parameters.Add(paramSSN);
sqlCmd.Parameters.Add(paramFirstName);
sqlCmd.Parameters.Add(paramLastName);
sqlCmd.Parameters.Add(paramBirthDate);

using (sqlCmd.Connection = new SqlConnection(connectionString))
{
    try
    {
        sqlCmd.Connection.Open();
        sqlCmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        returnValue = 1;
        Console.WriteLine("The following error was encountered: ");
        Console.WriteLine(ex.Message);
        Console.WriteLine(Environment.NewLine + "Press Enter key to exit");
        Console.ReadLine();
        Environment.Exit(0);
    }
}
return returnValue;
}

static List<Patient> SelectAllPatients()
{
    List<Patient> patients = new List<Patient>();

    SqlCommand sqlCmd = new SqlCommand(
        "SELECT [SSN], [FirstName], [LastName], [BirthDate] FROM [dbo].[Patients]",
        new SqlConnection(connectionString));

    using (sqlCmd.Connection = new SqlConnection(connectionString))

    using (sqlCmd.Connection = new SqlConnection(connectionString))
    {
        try
        {
            sqlCmd.Connection.Open();
            SqlDataReader reader = sqlCmd.ExecuteReader();

            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    patients.Add(new Patient()
                    {
                        SSN = reader[0].ToString(),
                        FirstName = reader[1].ToString(),
                        LastName = reader["LastName"].ToString(),
                        BirthDate = (DateTime)reader["BirthDate"]
                    });
                }
            }
        }
        catch (Exception ex)
        {

```

```

        throw;
    }
}

return patients;
}

static Patient SelectPatientBySSN(string ssn)
{
    Patient patient = new Patient();

    SqlCommand sqlCmd = new SqlCommand(
        "SELECT [SSN], [FirstName], [LastName], [BirthDate] FROM [dbo].[Patients] WHERE [SSN]=@SSN",
        new SqlConnection(connectionString);

    SqlParameter paramSSN = new SqlParameter("@@SSN", ssn);
    paramSSN.DbType = DbType.AnsiStringFixedLength;
    paramSSN.Direction = ParameterDirection.Input;
    paramSSN.Size = 11;

    sqlCmd.Parameters.Add(paramSSN);

    using (sqlCmd.Connection = new SqlConnection(connectionString))
    {
        try
        {
            sqlCmd.Connection.Open();
            SqlDataReader reader = sqlCmd.ExecuteReader();

            if (reader.HasRows)
            {
                while (reader.Read())
                {
                    patient = new Patient()
                    {
                        SSN = reader[0].ToString(),
                        FirstName = reader[1].ToString(),
                        LastName = reader["LastName"].ToString(),
                        BirthDate = (DateTime)reader["BirthDate"]
                    };
                }
            }
            else
            {
                patient = null;
            }
        }
        catch (Exception ex)
        {
            throw;
        }
    }
    return patient;
}

// This method simply deletes all records in the Patients table to reset our demo.
static int ResetPatientsTable()
{
    int returnValue = 0;

    SqlCommand sqlCmd = new SqlCommand("DELETE FROM Patients");
    using (sqlCmd.Connection = new SqlConnection(connectionString))
    {
        try
        {
            sqlCmd.Connection.Open();

```

```

        sqlCommand.Connection.Open();
        sqlCommand.ExecuteNonQuery();

    }
    catch (Exception ex)
    {
        returnValue = 1;
    }
}
return returnValue;
}

class Patient
{
    public string SSN { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
}
}

```

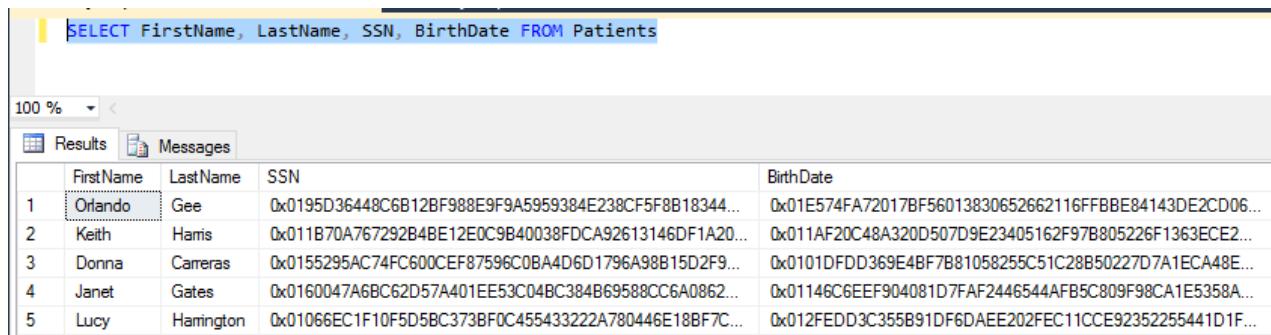
Verify that the data is encrypted

You can quickly check that the actual data on the server is encrypted by querying the **Patients** data with SSMS. (Use your current connection where the column encryption setting is not yet enabled.)

Run the following query on the Clinic database.

```
SELECT FirstName, LastName, SSN, BirthDate FROM Patients;
```

You can see that the encrypted columns do not contain any plaintext data.



The screenshot shows the SSMS interface with a query window containing the following SQL command:

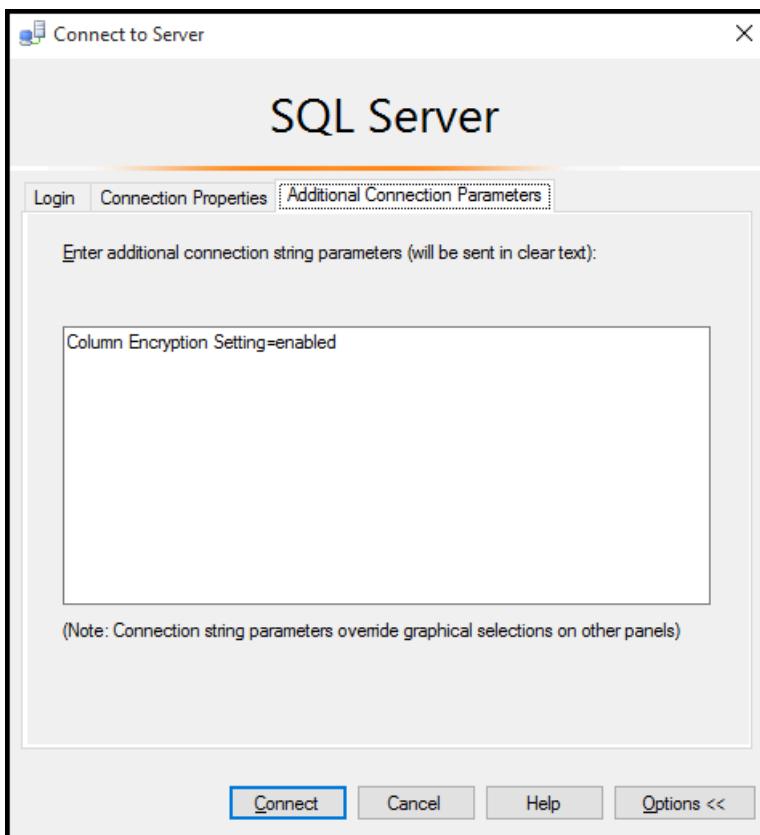
```
SELECT FirstName, LastName, SSN, BirthDate FROM Patients
```

The results grid displays the following data:

	FirstName	LastName	SSN	BirthDate
1	Orlando	Gee	0x0195D36448C6B12BF988E9F9A5959384E238CF5F8B18344...	0x01E574FA72017BF56013830652662116FFBBE84143DE2CD06...
2	Keith	Hamis	0x011B70A767292B4BE12E0C9B40038FDCA92613146DF1A20...	0x011AF20C48A320D507D9E23405162F97B805226F1363ECE2...
3	Donna	Carreras	0x0155295AC74FC600CEF87596C0BA4D6D1796A98B15D2F9...	0x0101DFDD369E4BF7B81058255C51C28B50227D7A1ECA48E...
4	Janet	Gates	0x0160047A6BC62D57A401EE53C04BC384B69588CC6A0862...	0x01146C6EF904081D7FAF2446544AFB5C809F98CA1E5358A...
5	Lucy	Hamington	0x01066EC1F10F5D5BC373BF0C45543322A780446E18BF7C...	0x012FEDD3C355B91DF6DAEE202FEC11CCE92352255441D1F...

To use SSMS to access the plaintext data, you can add the **Column Encryption Setting=enabled** parameter to the connection.

1. In SSMS, right-click your server in **Object Explorer**, and then click **Disconnect**.
2. Click **Connect > Database Engine** to open the **Connect to Server** window, and then click **Options**.
3. Click **Additional Connection Parameters** and type **Column Encryption Setting=enabled**.



- Run the following query on the **Clinic** database.

```
SELECT FirstName, LastName, SSN, BirthDate FROM Patients;
```

You can now see the plaintext data in the encrypted columns.

	FirstName	LastName	SSN	BirthDate
1	Orlando	Gee	999-99-0001	1964-01-04
2	Keith	Hamis	999-99-0002	1977-06-20
3	Donna	Carreras	999-99-0003	1973-02-09
4	Janet	Gates	999-99-0004	1985-08-31
5	Lucy	Harrington	999-99-0005	1993-05-06

NOTE

If you connect with SSMS (or any client) from a different computer, it will not have access to the encryption keys and will not be able to decrypt the data.

Next steps

After you create a database that uses Always Encrypted, you may want to do the following:

- Run this sample from a different computer. It won't have access to the encryption keys, so it will not have access to the plaintext data and will not run successfully.
- [Rotate and clean up your keys.](#)
- [Migrate data that is already encrypted with Always Encrypted.](#)
- [Deploy Always Encrypted certificates to other client machines](#) (see the "Making Certificates Available to

Applications and Users" section).

Related information

- [Always Encrypted \(client development\)](#)
- [Transparent Data Encryption](#)
- [SQL Server Encryption](#)
- [Always Encrypted Wizard](#)
- [Always Encrypted Blog](#)

Always Encrypted: Protect sensitive data and store encryption keys in Azure Key Vault

11/22/2019 • 14 minutes to read • [Edit Online](#)

This article shows you how to secure sensitive data in a SQL database with data encryption using the [Always Encrypted Wizard](#) in [SQL Server Management Studio \(SSMS\)](#). It also includes instructions that will show you how to store each encryption key in Azure Key Vault.

Always Encrypted is a new data encryption technology in Azure SQL Database and SQL Server that helps protect sensitive data at rest on the server, during movement between client and server, and while the data is in use. Always Encrypted ensures that sensitive data never appears as plaintext inside the database system. After you configure data encryption, only client applications or app servers that have access to the keys can access plaintext data. For detailed information, see [Always Encrypted \(Database Engine\)](#).

After you configure the database to use Always Encrypted, you will create a client application in C# with Visual Studio to work with the encrypted data.

Follow the steps in this article and learn how to set up Always Encrypted for an Azure SQL database. In this article you will learn how to perform the following tasks:

- Use the Always Encrypted wizard in SSMS to create [Always Encrypted keys](#).
 - Create a [column master key \(CMK\)](#).
 - Create a [column encryption key \(CEK\)](#).
- Create a database table and encrypt columns.
- Create an application that inserts, selects, and displays data from the encrypted columns.

Prerequisites

For this tutorial, you'll need:

- An Azure account and subscription. If you don't have one, sign up for a [free trial](#).
- [SQL Server Management Studio](#) version 13.0.700.242 or later.
- [.NET Framework 4.6](#) or later (on the client computer).
- [Visual Studio](#).
- [Azure PowerShell](#) or [Azure CLI](#)

Enable your client application to access the SQL Database service

You must enable your client application to access the SQL Database service by setting up an Azure Active Directory (AAD) application and copying the *Application ID* and *key* that you will need to authenticate your application.

To get the *Application ID* and *key*, follow the steps in [create an Azure Active Directory application and service principal that can access resources](#).

Create a key vault to store your keys

Now that your client app is configured and you have your application ID, it's time to create a key vault and configure its access policy so you and your application can access the vault's secrets (the Always Encrypted keys). The *create*, *get*, *list*, *sign*, *verify*, *wrapKey*, and *unwrapKey* permissions are required for creating a new column

master key and for setting up encryption with SQL Server Management Studio.

You can quickly create a key vault by running the following script. For a detailed explanation of these commands and more information about creating and configuring a key vault, see [What is Azure Key Vault?](#).

- [PowerShell](#)
- [Azure CLI](#)

IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

```
$subscriptionName = '<subscriptionName>'  
$userPrincipalName = '<username@domain.com>'  
$applicationId = '<applicationId from AAD application>'  
$resourceGroupName = '<resourceGroupName>' # use the same resource group name when creating your SQL Database below  
$location = '<datacenterLocation>'  
$vaultName = '<vaultName>'  
  
Connect-AzAccount  
$subscriptionId = (Get-AzSubscription -SubscriptionName $subscriptionName).Id  
Set-AzContext -SubscriptionId $subscriptionId  
  
New-AzResourceGroup -Name $resourceGroupName -Location $location  
New-AzKeyVault -VaultName $vaultName -ResourceGroupName $resourceGroupName -Location $location  
  
Set-AzKeyVaultAccessPolicy -VaultName $vaultName -ResourceGroupName $resourceGroupName -PermissionsToKeys  
create,get,wrapKey,unwrapKey,sign,verify,list -UserPrincipalName $userPrincipalName  
Set-AzKeyVaultAccessPolicy -VaultName $vaultName -ResourceGroupName $resourceGroupName -ServicePrincipalName  
$applicationId -PermissionsToKeys get,wrapKey,unwrapKey,sign,verify,list
```

Create a blank SQL database

1. Sign in to the [Azure portal](#).
2. Go to **Create a resource > Databases > SQL Database**.
3. Create a **Blank** database named **Clinic** on a new or existing server. For detailed directions about how to create a database in the Azure portal, see [Your first Azure SQL database](#).

Create SQL Database

Microsoft

Basics • [Additional settings](#) [Review + Create](#)

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription [?](#) [\[Subscription\]](#) ▾
 └─ * Resource group [?](#) [\[Resource group\]](#) ▾
[Create new](#)

INSTANCE DETAILS

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

* Database name [Enter database name](#)
 * Server [Select a server](#) ▾
[Create new](#)
 The value should not be empty.
 Yes No

* Want to use SQL elastic pool? [?](#)
 Yes No

* Compute + storage [?](#) [Please select a server first.](#)
[Configure database](#)

[Review + Create](#) [Next : Additional settings >>](#)

You will need the connection string later in the tutorial, so after you create the database, browse to the new Clinic database and copy the connection string. You can get the connection string at any time, but it's easy to copy it in the Azure portal.

1. Go to **SQL databases** > **Clinic** > **Show database connection strings**.
2. Copy the connection string for **ADO.NET**.

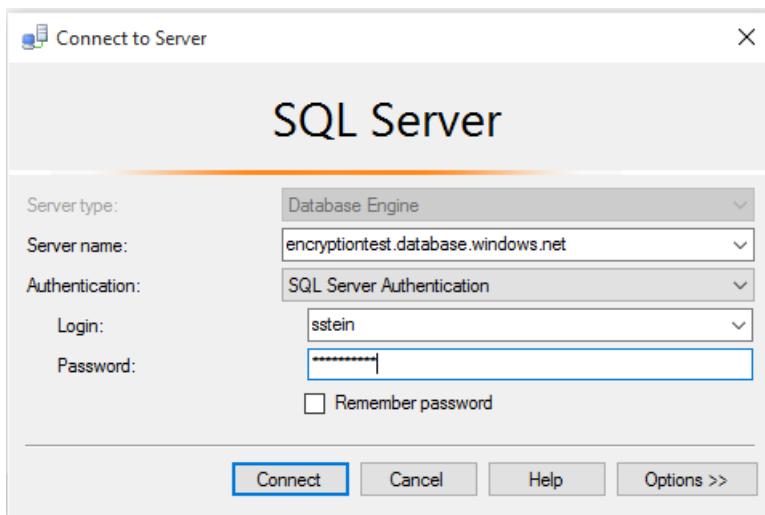


Connect to the database with SSMS

Open SSMS and connect to the server with the Clinic database.

1. Open SSMS. (Go to **Connect** > **Database Engine** to open the **Connect to Server** window if it isn't open.)

2. Enter your server name and credentials. The server name can be found on the SQL database blade and in the connection string you copied earlier. Type the complete server name, including *database.windows.net*.



If the **New Firewall Rule** window opens, sign in to Azure and let SSMS create a new firewall rule for you.

Create a table

In this section, you will create a table to hold patient data. It's not initially encrypted--you will configure encryption in the next section.

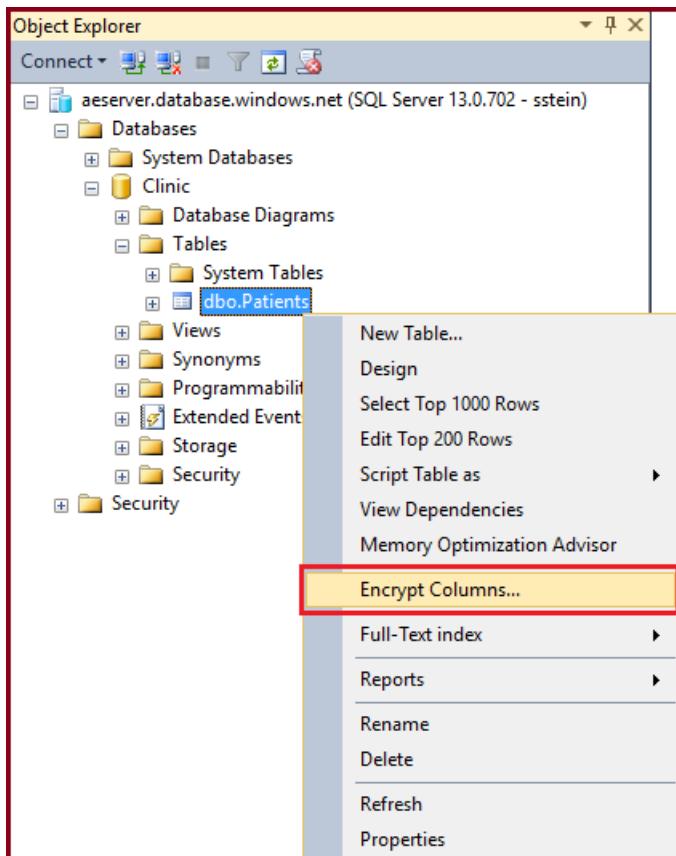
1. Expand **Databases**.
2. Right-click the **Clinic** database and click **New Query**.
3. Paste the following Transact-SQL (T-SQL) into the new query window and **Execute** it.

```
CREATE TABLE [dbo].[Patients](
    [PatientId] [int] IDENTITY(1,1),
    [SSN] [char](11) NOT NULL,
    [FirstName] [nvarchar](50) NULL,
    [LastName] [nvarchar](50) NULL,
    [MiddleName] [nvarchar](50) NULL,
    [StreetAddress] [nvarchar](50) NULL,
    [City] [nvarchar](50) NULL,
    [ZipCode] [char](5) NULL,
    [State] [char](2) NULL,
    [BirthDate] [date] NOT NULL
    PRIMARY KEY CLUSTERED ([PatientId] ASC) ON [PRIMARY] );
GO
```

Encrypt columns (configure Always Encrypted)

SSMS provides a wizard that helps you easily configure Always Encrypted by setting up the column master key, column encryption key, and encrypted columns for you.

1. Expand **Databases** > **Clinic** > **Tables**.
2. Right-click the **Patients** table and select **Encrypt Columns** to open the Always Encrypted wizard:



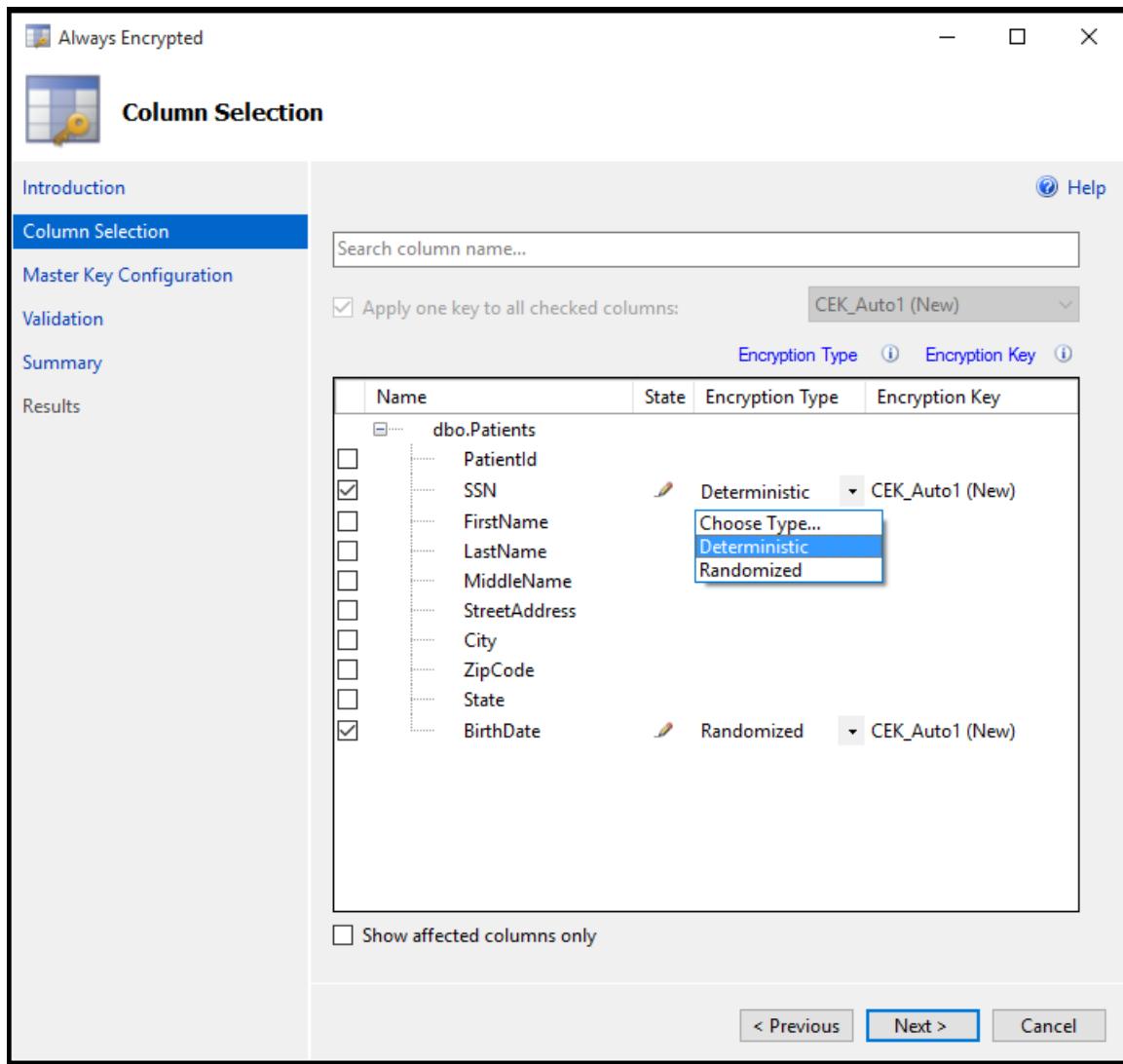
The Always Encrypted wizard includes the following sections: **Column Selection**, **Master Key Configuration**, **Validation**, and **Summary**.

Column Selection

Click **Next** on the **Introduction** page to open the **Column Selection** page. On this page, you will select which columns you want to encrypt, [the type of encryption](#), and [what column encryption key \(CEK\)](#) to use.

Encrypt **SSN** and **BirthDate** information for each patient. The SSN column will use deterministic encryption, which supports equality lookups, joins, and group by. The BirthDate column will use randomized encryption, which does not support operations.

Set the **Encryption Type** for the SSN column to **Deterministic** and the BirthDate column to **Randomized**. Click **Next**.

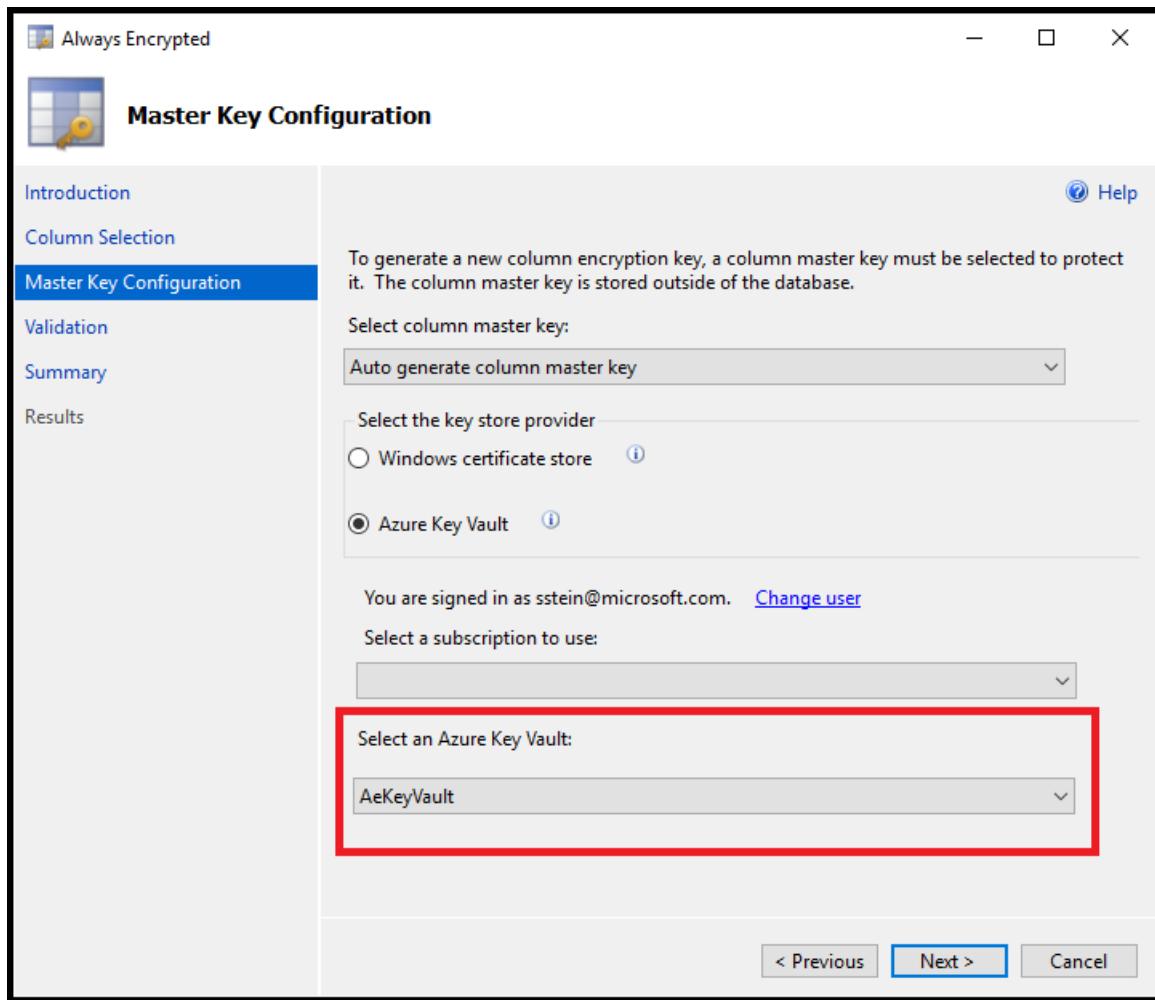


Master Key Configuration

The **Master Key Configuration** page is where you set up your CMK and select the key store provider where the CMK will be stored. Currently, you can store a CMK in the Windows certificate store, Azure Key Vault, or a hardware security module (HSM).

This tutorial shows how to store your keys in Azure Key Vault.

1. Select **Azure Key Vault**.
2. Select the desired key vault from the drop-down list.
3. Click **Next**.

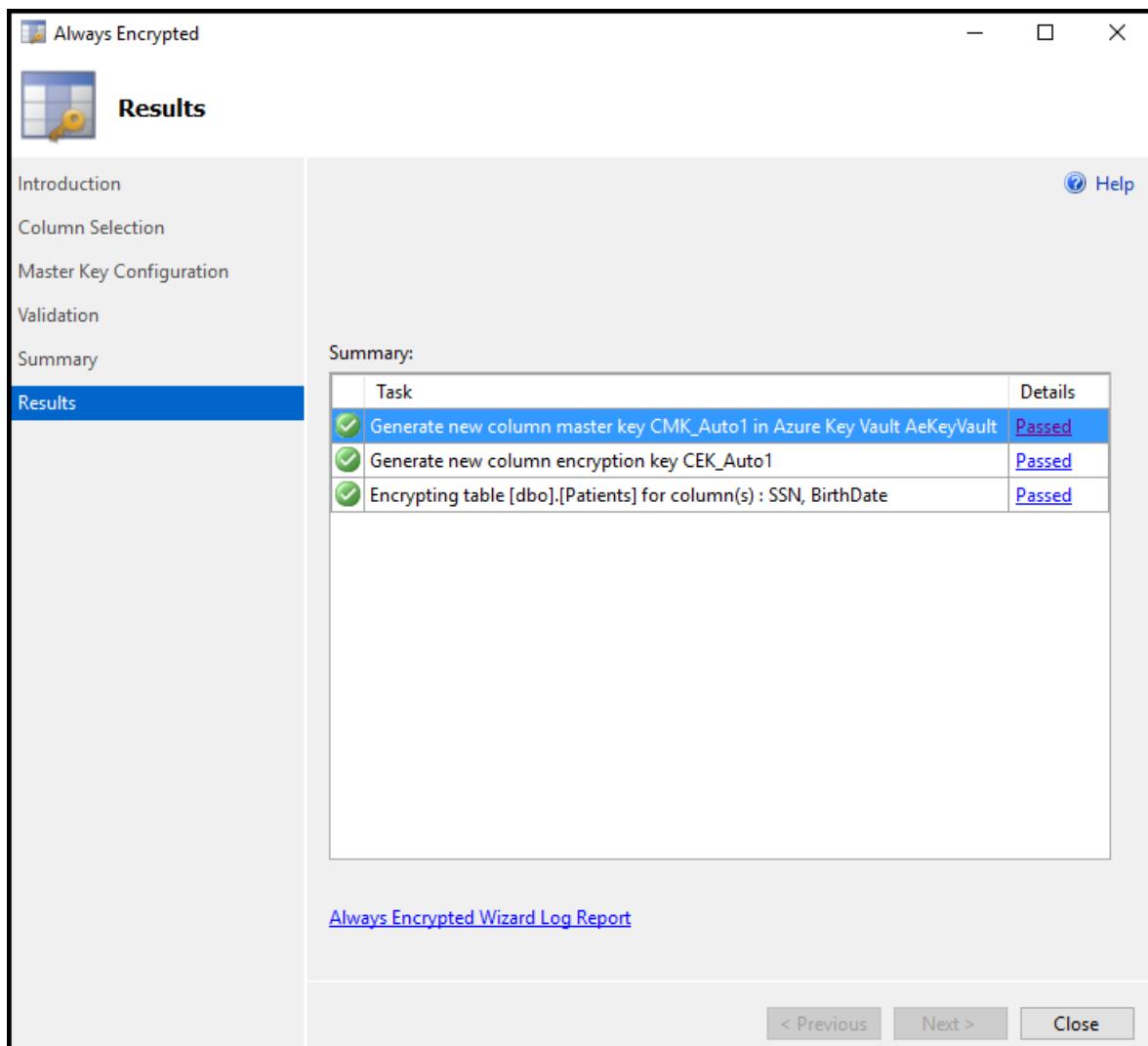


Validation

You can encrypt the columns now or save a PowerShell script to run later. For this tutorial, select **Proceed to finish now** and click **Next**.

Summary

Verify that the settings are all correct and click **Finish** to complete the setup for Always Encrypted.



Verify the wizard's actions

After the wizard is finished, your database is set up for Always Encrypted. The wizard performed the following actions:

- Created a column master key and stored it in Azure Key Vault.
- Created a column encryption key and stored it in Azure Key Vault.
- Configured the selected columns for encryption. The Patients table currently has no data, but any existing data in the selected columns is now encrypted.

You can verify the creation of the keys in SSMS by expanding **Clinic > Security > Always Encrypted Keys**.

Create a client application that works with the encrypted data

Now that Always Encrypted is set up, you can build an application that performs *inserts* and *selects* on the encrypted columns.

IMPORTANT

Your application must use **SqlParameter** objects when passing plaintext data to the server with Always Encrypted columns. Passing literal values without using SqlParameter objects will result in an exception.

1. Open Visual Studio and create a new C# **Console Application** (Visual Studio 2015 and earlier) or **Console App (.NET Framework)** (Visual Studio 2017 and later). Make sure your project is set to **.NET Framework 4.6** or later.
2. Name the project **AlwaysEncryptedConsoleAKVApp** and click **OK**.

3. Install the following NuGet packages by going to **Tools > NuGet Package Manager > Package Manager Console**.

Run these two lines of code in the Package Manager Console:

```
Install-Package Microsoft.SqlServer.Management.AlwaysEncrypted.AzureKeyVaultProvider  
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

Modify your connection string to enable Always Encrypted

This section explains how to enable Always Encrypted in your database connection string.

To enable Always Encrypted, you need to add the **Column Encryption Setting** keyword to your connection string and set it to **Enabled**.

You can set this directly in the connection string, or you can set it by using [SqlConnectionStringBuilder](#). The sample application in the next section shows how to use [SqlConnectionStringBuilder](#).

Enable Always Encrypted in the connection string

Add the following keyword to your connection string.

```
Column Encryption Setting=Enabled
```

Enable Always Encrypted with SqlConnectionStringBuilder

The following code shows how to enable Always Encrypted by setting [SqlConnectionStringBuilder.ColumnEncryptionSetting](#) to [Enabled](#).

```
// Instantiate a SqlConnectionStringBuilder.  
SqlConnectionStringBuilder connStringBuilder = new SqlConnectionStringBuilder("replace with your connection  
string");  
  
// Enable Always Encrypted.  
connStringBuilder.ColumnEncryptionSetting = SqlConnectionColumnEncryptionSetting.Enabled;
```

Register the Azure Key Vault provider

The following code shows how to register the Azure Key Vault provider with the ADO.NET driver.

```
private static ClientCredential _clientCredential;  
  
static void InitializeAzureKeyVaultProvider() {  
    _clientCredential = new ClientCredential(applicationId, clientKey);  
  
    SqlColumnEncryptionAzureKeyVaultProvider azureKeyVaultProvider = new  
    SqlColumnEncryptionAzureKeyVaultProvider(GetToken);  
  
    Dictionary<string, SqlColumnEncryptionKeyStoreProvider> providers = new Dictionary<string,  
    SqlColumnEncryptionKeyStoreProvider>();  
  
    providers.Add(SqlColumnEncryptionAzureKeyVaultProvider.ProviderName, azureKeyVaultProvider);  
    SqlConnection.RegisterColumnEncryptionKeyStoreProviders(providers);  
}
```

Always Encrypted sample console application

This sample demonstrates how to:

- Modify your connection string to enable Always Encrypted.
- Register Azure Key Vault as the application's key store provider.
- Insert data into the encrypted columns.
- Select a record by filtering for a specific value in an encrypted column.

Replace the contents of *Program.cs* with the following code. Replace the connection string for the global `connectionString` variable in the line that directly precedes the `Main` method with your valid connection string from the Azure portal. This is the only change you need to make to this code.

Run the app to see Always Encrypted in action.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.SqlServer.Management.AlwaysEncrypted.AzureKeyVaultProvider;

namespace AlwaysEncryptedConsoleAKVApp {
    class Program {
        // Update this line with your Clinic database connection string from the Azure portal.
        static string connectionString = @"<connection string from the portal>";
        static string applicationId = @"<application ID from your AAD application>";
        static string clientKey = "<key from your AAD application>";

        static void Main(string[] args) {
            InitializeAzureKeyVaultProvider();

            Console.WriteLine("Signed in as: " + _clientCredential.ClientId);

            Console.WriteLine("Original connection string copied from the Azure portal:");
            Console.WriteLine(connectionString);

            // Create a SqlConnectionStringBuilder.
            SqlConnectionStringBuilder connStringBuilder =
                new SqlConnectionStringBuilder(connectionString);

            // Enable Always Encrypted for the connection.
            // This is the only change specific to Always Encrypted
            connStringBuilder.ColumnEncryptionSetting =
                SqlConnectionColumnEncryptionSetting.Enabled;

            Console.WriteLine(Environment.NewLine + "Updated connection string with Always Encrypted
enabled:");
            Console.WriteLine(connStringBuilder.ConnectionString);

            // Update the connection string with a password supplied at runtime.
            Console.WriteLine(Environment.NewLine + "Enter server password:");
            connStringBuilder.Password = Console.ReadLine();

            // Assign the updated connection string to our global variable.
            connectionString = connStringBuilder.ConnectionString;

            // Delete all records to restart this demo app.
            ResetPatientsTable();

            // Add sample data to the Patients table.
            Console.WriteLine(Environment.NewLine + "Adding sample patient data to the database...");

            InsertPatient(new Patient() {
                SSN = "999-99-0001",
                FirstName = "Orlando",
                LastName = "Goo"
            });
        }
    }
}
```

```

    LastName = "Gee",
    BirthDate = DateTime.Parse("01/04/1964")
});
InsertPatient(new Patient() {
    SSN = "999-99-0002",
    FirstName = "Keith",
    LastName = "Harris",
    BirthDate = DateTime.Parse("06/20/1977")
});
InsertPatient(new Patient() {
    SSN = "999-99-0003",
    FirstName = "Donna",
    LastName = "Carreras",
    BirthDate = DateTime.Parse("02/09/1973")
});
InsertPatient(new Patient() {
    SSN = "999-99-0004",
    FirstName = "Janet",
    LastName = "Gates",
    BirthDate = DateTime.Parse("08/31/1985")
});
InsertPatient(new Patient() {
    SSN = "999-99-0005",
    FirstName = "Lucy",
    LastName = "Harrington",
    BirthDate = DateTime.Parse("05/06/1993")
});

// Fetch and display all patients.
Console.WriteLine(Environment.NewLine + "All the records currently in the Patients table:");

foreach (Patient patient in SelectAllPatients()) {
    Console.WriteLine(patient.FirstName + " " + patient.LastName + "\tSSN: " + patient.SSN +
"\tBirthdate: " + patient.BirthDate);
}

// Get patients by SSN.
Console.WriteLine(Environment.NewLine + "Now lets locate records by searching the encrypted SSN
column.");
string ssn;

// This very simple validation only checks that the user entered 11 characters.
// In production be sure to check all user input and use the best validation for your specific
application.
do {
    Console.WriteLine("Please enter a valid SSN (ex. 999-99-0003):");
    ssn = Console.ReadLine();
} while (ssn.Length != 11);

// The example allows duplicate SSN entries so we will return all records
// that match the provided value and store the results in selectedPatients.
Patient selectedPatient = SelectPatientBySSN(ssn);

// Check if any records were returned and display our query results.
if (selectedPatient != null) {
    Console.WriteLine("Patient found with SSN = " + ssn);
    Console.WriteLine(selectedPatient.FirstName + " " + selectedPatient.LastName + "\tSSN: "
+ selectedPatient.SSN + "\tBirthdate: " + selectedPatient.BirthDate);
}
else {
    Console.WriteLine("No patients found with SSN = " + ssn);
}

Console.WriteLine("Press Enter to exit...");
Console.ReadLine();
}

private static ClientCredential _clientCredential;

```

```

static void InitializeAzureKeyVaultProvider() {
    _clientCredential = new ClientCredential(applicationId, clientKey);

    SqlColumnEncryptionAzureKeyVaultProvider azureKeyVaultProvider =
        new SqlColumnEncryptionAzureKeyVaultProvider(GetToken);

    Dictionary<string, SqlColumnEncryptionKeyStoreProvider> providers =
        new Dictionary<string, SqlColumnEncryptionKeyStoreProvider>();

    providers.Add(SqlColumnEncryptionAzureKeyVaultProvider.ProviderName, azureKeyVaultProvider);
    SqlConnection.RegisterColumnEncryptionKeyStoreProviders(providers);
}

public async static Task<string> GetToken(string authority, string resource, string scope) {
    var authContext = new AuthenticationContext(authority);
    AuthenticationResult result = await authContext.AcquireTokenAsync(resource, _clientCredential);

    if (result == null)
        throw new InvalidOperationException("Failed to obtain the access token");
    return result.AccessToken;
}

static int InsertPatient(Patient newPatient) {
    int returnValue = 0;

    string sqlCmdText = @"INSERT INTO [dbo].[Patients] ([SSN], [FirstName], [LastName], [BirthDate])
VALUES (@SSN, @FirstName, @LastName, @BirthDate);";

    SqlCommand sqlCmd = new SqlCommand(sqlCmdText);

    SqlParameter paramSSN = new SqlParameter("@@SSN", newPatient.SSN);
    paramSSN.DbType = DbType.AnsiStringFixedLength;
    paramSSN.Direction = ParameterDirection.Input;
    paramSSN.Size = 11;

    SqlParameter paramFirstName = new SqlParameter("@@FirstName", newPatient.FirstName);
    paramFirstName.DbType = DbType.String;
    paramFirstName.Direction = ParameterDirection.Input;

    SqlParameter paramLastName = new SqlParameter("@@LastName", newPatient.LastName);
    paramLastName.DbType = DbType.String;
    paramLastName.Direction = ParameterDirection.Input;

    SqlParameter paramBirthDate = new SqlParameter("@@BirthDate", newPatient.BirthDate);
    paramBirthDate.SqlDbType = SqlDbType.Date;
    paramBirthDate.Direction = ParameterDirection.Input;

    sqlCmd.Parameters.Add(paramSSN);
    sqlCmd.Parameters.Add(paramFirstName);
    sqlCmd.Parameters.Add(paramLastName);
    sqlCmd.Parameters.Add(paramBirthDate);

    using (sqlCmd.Connection = new SqlConnection(connectionString)) {
        try {
            sqlCmd.Connection.Open();
            sqlCmd.ExecuteNonQuery();
        }
        catch (Exception ex) {
            returnValue = 1;
            Console.WriteLine("The following error was encountered: ");
            Console.WriteLine(ex.Message);
            Console.WriteLine(Environment.NewLine + "Press Enter key to exit");
            Console.ReadLine();
            Environment.Exit(0);
        }
    }
    return returnValue;
}

```

```

static List<Patient> SelectAllPatients() {
    List<Patient> patients = new List<Patient>();

    SqlCommand sqlCmd = new SqlCommand(
        "SELECT [SSN], [FirstName], [LastName], [BirthDate] FROM [dbo].[Patients]",
        new SqlConnection(connectionString));

    using (sqlCmd.Connection = new SqlConnection(connectionString))

    using (sqlCmd.Connection = new SqlConnection(connectionString)) {
        try {
            sqlCmd.Connection.Open();
            SqlDataReader reader = sqlCmd.ExecuteReader();

            if (reader.HasRows) {
                while (reader.Read()) {
                    patients.Add(new Patient() {
                        SSN = reader[0].ToString(),
                        FirstName = reader[1].ToString(),
                        LastName = reader["LastName"].ToString(),
                        BirthDate = (DateTime)reader["BirthDate"]
                    });
                }
            }
        catch (Exception ex) {
            throw;
        }
    }

    return patients;
}

static Patient SelectPatientBySSN(string ssn) {
    Patient patient = new Patient();

    SqlCommand sqlCmd = new SqlCommand(
        "SELECT [SSN], [FirstName], [LastName], [BirthDate] FROM [dbo].[Patients] WHERE [SSN]=@SSN",
        new SqlConnection(connectionString);

    SqlParameter paramSSN = new SqlParameter("@@SSN", ssn);
    paramSSN.DbType = DbType.AnsiStringFixedLength;
    paramSSN.Direction = ParameterDirection.Input;
    paramSSN.Size = 11;

    sqlCmd.Parameters.Add(paramSSN);

    using (sqlCmd.Connection = new SqlConnection(connectionString)) {
        try {
            sqlCmd.Connection.Open();
            SqlDataReader reader = sqlCmd.ExecuteReader();

            if (reader.HasRows) {
                while (reader.Read()) {
                    patient = new Patient() {
                        SSN = reader[0].ToString(),
                        FirstName = reader[1].ToString(),
                        LastName = reader["LastName"].ToString(),
                        BirthDate = (DateTime)reader["BirthDate"]
                    };
                }
            }
            else {
                patient = null;
            }
        }
        catch (Exception ex) {

```

```

        throw;
    }
}

return patient;
}

// This method simply deletes all records in the Patients table to reset our demo.
static int ResetPatientsTable() {
    int returnValue = 0;

    SqlCommand sqlCmd = new SqlCommand("DELETE FROM Patients");
    using (sqlCmd.Connection = new SqlConnection(connectionString)) {
        try {
            sqlCmd.Connection.Open();
            sqlCmd.ExecuteNonQuery();

        }
        catch (Exception ex) {
            returnValue = 1;
        }
    }
    return returnValue;
}
}

class Patient {
    public string SSN { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
}
}

```

Verify that the data is encrypted

You can quickly check that the actual data on the server is encrypted by querying the Patients data with SSMS (using your current connection where **Column Encryption Setting** is not yet enabled).

Run the following query on the Clinic database.

```
SELECT FirstName, LastName, SSN, BirthDate FROM Patients;
```

You can see that the encrypted columns do not contain any plaintext data.

	FirstName	LastName	SSN	BirthDate
1	Orlando	Gee	0x0195D36448C6B12BF988E9F9A5959384E238CF5F8B18344...	0x01E574FA72017BF56013830652662116FFBBE84143DE2CD06...
2	Keith	Hamis	0x011B70A767292B4BE12E0C9B40038FDCA92613146DF1A20...	0x011AF20C48A320D507D9E23405162F97B805226F1363ECE2...
3	Donna	Carrera	0x0155295AC74FC600CEF87596C0BA4D6D1796A98B15D2F9...	0x0101DFDD369E4BF7B81058255C51C28B50227D7A1ECA48E...
4	Janet	Gates	0x0160047A6BC62D57A401EE53C04BC384B69588CC6A0862...	0x01146C6EEF904081D7FAF2446544AFB5C809F98CA1E5358A...
5	Lucy	Hamington	0x01066EC1F10F5D5BC373BF0C45543322A780446E18BF7C...	0x012FEDD3C355B91DF6DAEE202FEC11CCE92352255441D1F...

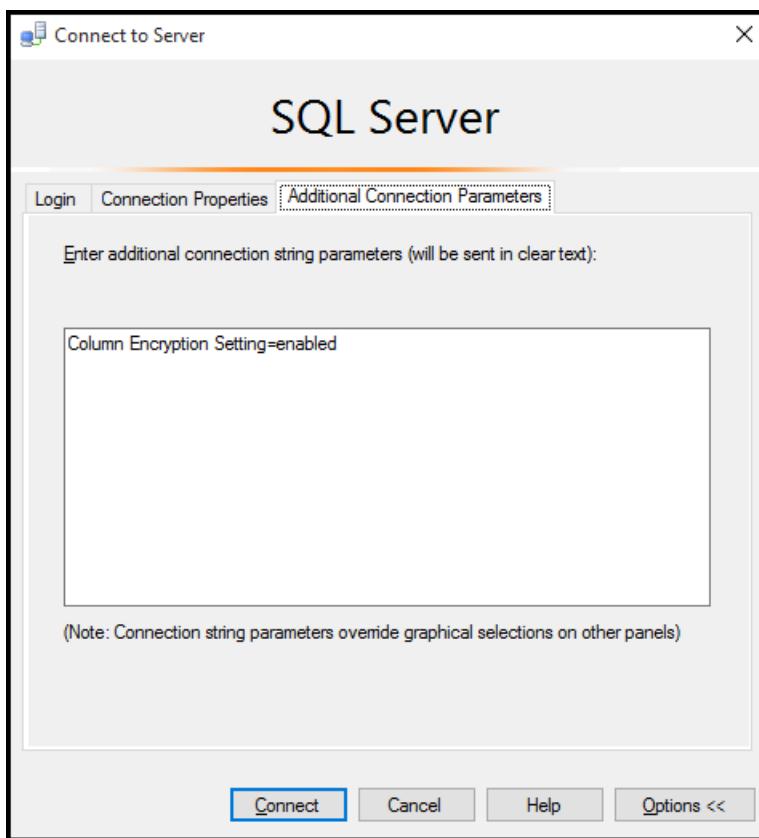
To use SSMS to access the plaintext data, you first need to ensure that the user has proper permissions to the Azure Key Vault: *get*, *unwrapKey*, and *verify*. For detailed information, see [Create and Store Column Master Keys \(Always Encrypted\)](#).

Then add the *Column Encryption Setting=enabled* parameter during your connection.

1. In SSMS, right-click your server in **Object Explorer** and choose **Disconnect**.

2. Click **Connect > Database Engine** to open the **Connect to Server** window and click **Options**.

3. Click **Additional Connection Parameters** and type **Column Encryption Setting=enabled**.



4. Run the following query on the Clinic database.

```
SELECT FirstName, LastName, SSN, BirthDate FROM Patients;
```

You can now see the plaintext data in the encrypted columns.

	FirstName	LastName	SSN	BirthDate
1	Orlando	Gee	999-99-0001	1964-01-04
2	Keith	Haris	999-99-0002	1977-06-20
3	Donna	Carreras	999-99-0003	1973-02-09
4	Janet	Gates	999-99-0004	1985-08-31
5	Lucy	Hamington	999-99-0005	1993-05-06

Next steps

After you create a database that uses Always Encrypted, you may want to do the following:

- [Rotate and clean up your keys.](#)
- [Migrate data that is already encrypted with Always Encrypted.](#)

Related information

- [Always Encrypted \(client development\)](#)
- [Transparent data encryption](#)
- [SQL Server encryption](#)

- [Always Encrypted wizard](#)
- [Always Encrypted blog](#)

Get started with SQL database auditing

2/14/2020 • 13 minutes to read • [Edit Online](#)

Auditing for Azure [SQL Database](#) and [SQL Data Warehouse](#) tracks database events and writes them to an audit log in your Azure storage account, Log Analytics workspace or Event Hubs. Auditing also:

- Helps you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations.
- Enables and facilitates adherence to compliance standards, although it doesn't guarantee compliance. For more information about Azure programs that support standards compliance, see the [Azure Trust Center](#) where you can find the most current list of SQL Database compliance certifications.

NOTE

This topic applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

Azure SQL database auditing overview

You can use SQL database auditing to:

- **Retain** an audit trail of selected events. You can define categories of database actions to be audited.
- **Report** on database activity. You can use pre-configured reports and a dashboard to get started quickly with activity and event reporting.
- **Analyze** reports. You can find suspicious events, unusual activity, and trends.

IMPORTANT

Audit logs are written to **Append Blobs** in Azure Blob storage on your Azure subscription.

- All storage kinds (v1, v2, blob) are supported.
- All storage replication configurations are supported.
- Storage behind a virtual network and firewall is supported.
- **Premium storage** is currently **not supported**.
- **Hierarchical namespace** for **Azure Data Lake Storage Gen2 storage account** is currently **not supported**.
- Enabling auditing on a paused **Azure SQL Data Warehouse** is not supported. To enable auditing, resume the Data Warehouse.

Define server-level vs. database-level auditing policy

An auditing policy can be defined for a specific database or as a default server policy:

- A server policy applies to all existing and newly created databases on the server.
- If *server blob auditing is enabled*, it *always applies to the database*. The database will be audited, regardless of the database auditing settings.
- Enabling blob auditing on the database or data warehouse, in addition to enabling it on the server, does *not* override or change any of the settings of the server blob auditing. Both audits will exist side

by side. In other words, the database is audited twice in parallel; once by the server policy and once by the database policy.

NOTE

You should avoid enabling both server blob auditing and database blob auditing together, unless:

- You want to use a different *storage account* or *retention period* for a specific database.
- You want to audit event types or categories for a specific database that differ from the rest of the databases on the server. For example, you might have table inserts that need to be audited only for a specific database.

Otherwise, we recommend that you enable only server-level blob auditing and leave the database-level auditing disabled for all databases.

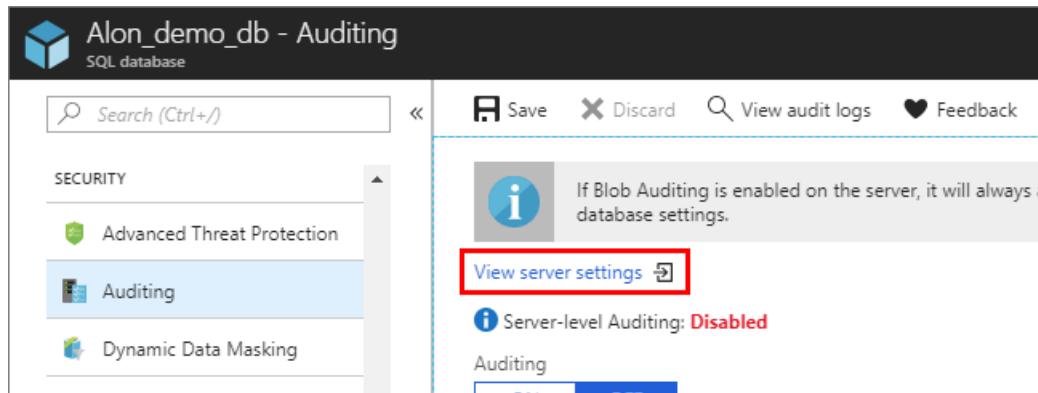
Set up auditing for your server

The following section describes the configuration of auditing using the Azure portal.

NOTE

You now have multiple options for configuring where audit logs are written. You can write logs to an Azure storage account, to a Log Analytics workspace for consumption by Azure Monitor logs, or to event hub for consumption using event hub. You can configure any combination of these options, and audit logs will be written to each.

1. Go to the [Azure portal](#).
2. Navigate to **Auditing** under the Security heading in your SQL database/server pane.
3. If you prefer to set up a server auditing policy, you can select the **View server settings** link on the database auditing page. You can then view or modify the server auditing settings. Server auditing policies apply to all existing and newly created databases on this server.



4. If you prefer to enable auditing on the database level, switch **Auditing** to **ON**. If server auditing is enabled, the database-configured audit will exist side-by-side with the server audit.

The screenshot shows the Azure portal interface for managing auditing settings. On the left, there's a sidebar with a search bar and a list of items including 'Automation script', 'SECURITY', 'Advanced Threat Protection', and 'Auditing'. The 'Auditing' item is selected and highlighted with a blue background. The main content area has a header with 'Save', 'Discard', 'View audit logs', and 'Feedback' buttons. Below the header, there's an information icon with a message: 'If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.' A 'View server settings' link is provided. Under the 'Auditing' section, it says 'Server-level Auditing: Disabled'. A large red box highlights the 'Auditing' section, specifically the 'ON' button, which is currently selected.

To configure writing audit logs to a storage account, select **Storage** and open **Storage details**. Select the Azure storage account where logs will be saved, and then select the retention period. Then click **OK**. Logs older than the retention period are deleted.

IMPORTANT

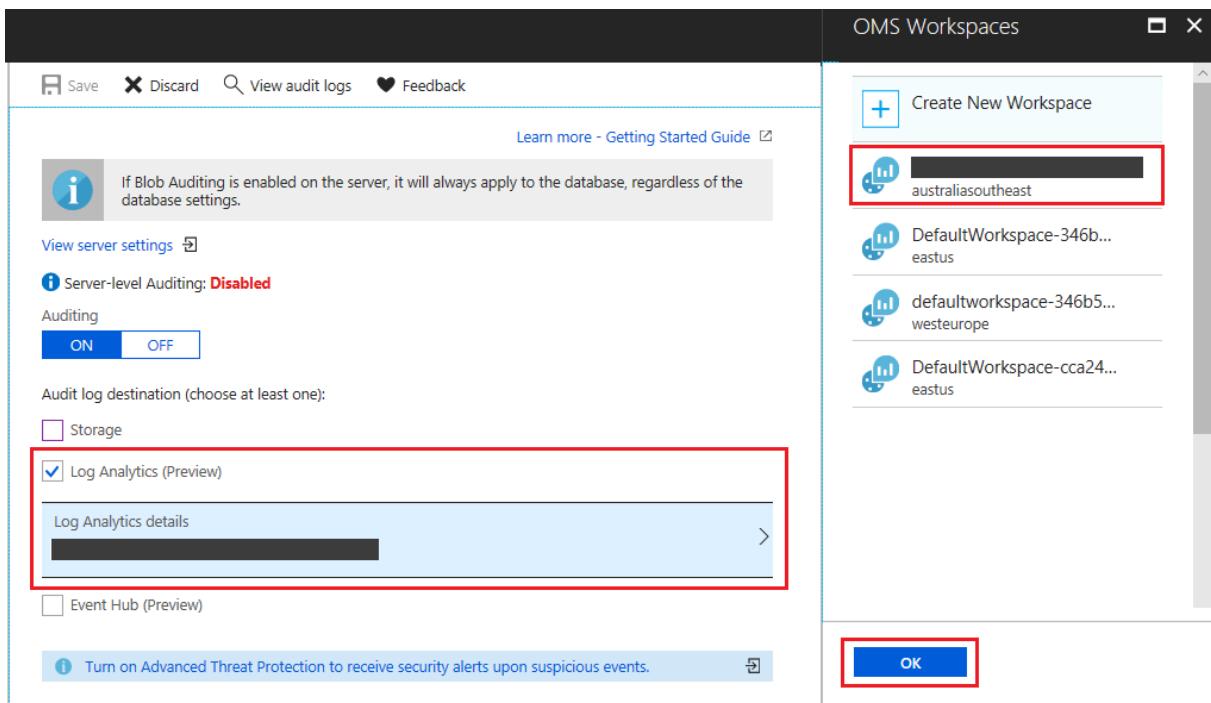
- The default value for retention period is 0 (unlimited retention). You can change this value by moving the **Retention (Days)** slider in **Storage settings** when configuring the storage account for auditing.
- If you change retention period from 0 (unlimited retention) to any other value, please note that retention will only apply to logs written after retention value was changed (logs written during the period when retention was set to unlimited are preserved, even after retention is enabled)

The screenshot shows the 'Storage settings' dialog box. At the top, there are 'Save', 'Discard', 'View audit logs', and 'Feedback' buttons. Below that, there's an information icon with a message: 'If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.' A 'View server settings' link is present. Under the 'Auditing' section, it says 'Server-level Auditing: Disabled'. A red box highlights the 'Auditing' section, specifically the 'ON' button, which is currently selected. In the 'Audit log destination' section, there are checkboxes for 'Storage', 'Log Analytics (Preview)', and 'Event Hub (Preview)'. The 'Storage' checkbox is checked and highlighted with a red box. Below it is a 'Storage details' button, also highlighted with a red box. To the right, the 'Storage settings' pane is visible, containing fields for 'Subscription' (highlighted with a red box), 'Storage account' (highlighted with a red box), and 'Retention (Days)' (highlighted with a red box). The 'Retention (Days)' slider is set to 0. At the bottom right of the dialog, there's a 'Storage access key' section with 'Primary' and 'Secondary' buttons, and a large red box highlights the 'OK' button.

To configure a storage account under a virtual network or firewall you will need an [Active Directory admin](#) on the server, enable **Allow trusted Microsoft services to access this storage account** on the storage account. In addition, you need to have the 'Microsoft.Authorization/roleAssignments/write' permission on the selected storage account.

We recommend you to be [User Access Administrator](#) in order to grant to the managed identity the role 'storage blob data contributor'. To learn more about permissions and role-based access control, see [What is role-based access control \(RBAC\) for Azure resources?](#) and [Add or remove role assignments using Azure RBAC and the Azure portal](#)

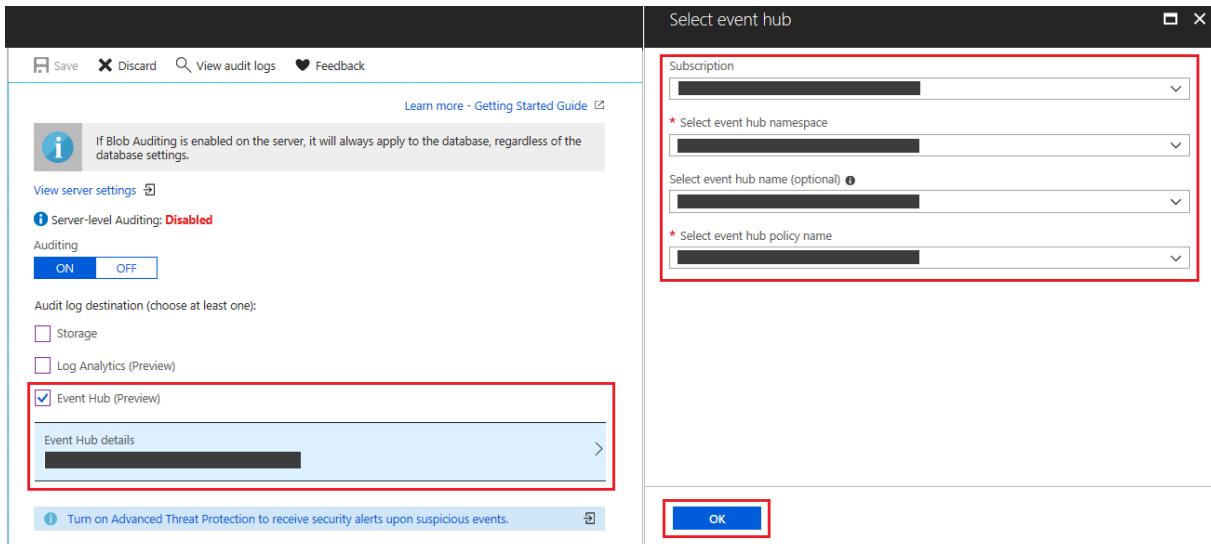
To configure writing audit logs to a Log Analytics workspace, select **Log Analytics (Preview)** and open **Log Analytics details**. Select or create the Log Analytics workspace where logs will be written and then click **OK**.



WARNING

Enabling auditing to Log Analytics will incur cost based on ingestion rates. Please be aware of the associated cost with using this [option](#), or consider storing the audit logs in an Azure storage account.

To configure writing audit logs to an event hub, select **Event Hub (Preview)** and open **Event Hub details**. Select the event hub where logs will be written and then click **OK**. Be sure that the event hub is in the same region as your database and server.



Analyze audit logs and reports

If you chose to write audit logs to Azure Monitor logs:

- Use the [Azure portal](#). Open the relevant database. At the top of the database's **Auditing** page, click **View audit logs**.

The screenshot shows the Azure portal interface for managing auditing settings. In the top navigation bar, the database name is displayed as '[-demo-db] - demo-server/-demo-db - Auditing'. Below the navigation bar, there is a search bar and several action buttons: 'Save', 'Discard', 'View audit logs' (which is highlighted with a red box), and 'Feedback'. On the left, a sidebar menu includes 'Security' (with 'Advanced Data Security', 'Auditing' selected, and 'Dynamic Data Masking', 'Transparent data encryption'), 'Intelligent Performance' (with 'Performance overview', 'Performance recommendation...', 'Query Performance Insight', and 'Automatic tuning'). The main content area is titled 'Auditing' and shows that 'Server-level Auditing' is 'Enabled'. A toggle switch is set to 'ON'. Under 'Audit log destination (choose at least one)', the 'Log Analytics (Preview)' checkbox is checked. A note states: 'If Blob Auditing is enabled on the server, it will always apply to the database, regardless of database settings.' A link to 'View server settings' is also present.

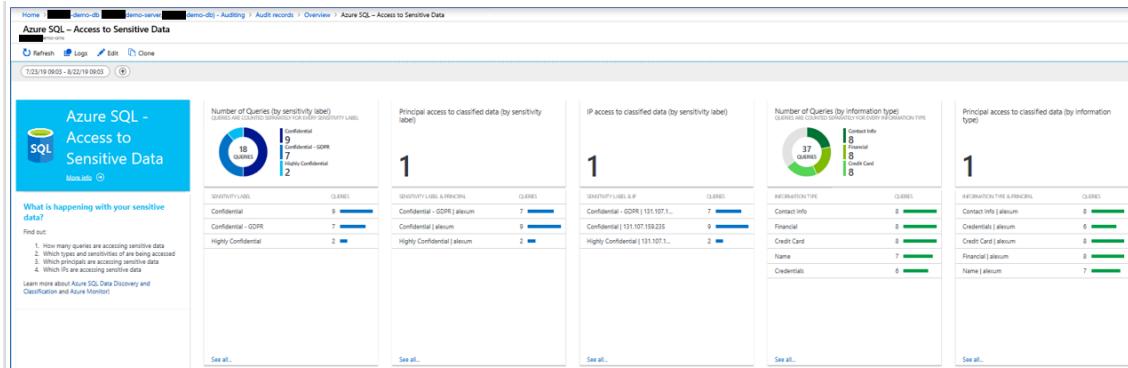
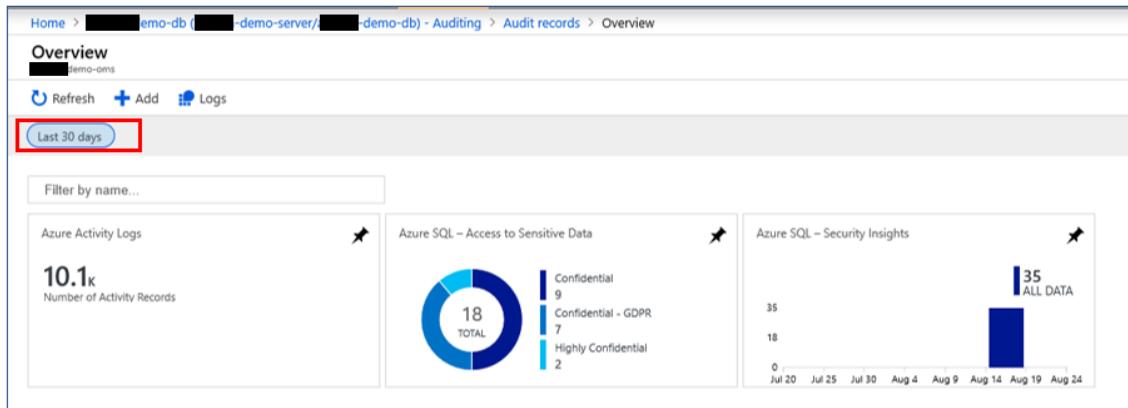
- Then, you have two ways to view the logs:

Clicking on **Log Analytics** at the top of the **Audit records** page will open the Logs view in Log Analytics workspace, where you can customize the time range and the search query.

The screenshot shows the 'Audit records' page. The top navigation bar includes 'Home', 'SQL databases', the database name, and 'Audit records'. Below the navigation bar are buttons for 'Refresh', 'Filter', 'Log Analytics' (which is highlighted with a red box), and 'View dashboard'. A note below the buttons says: 'Click here to learn more about alternative methods for viewing & analyzing audit records.' Underneath, there are sections for 'Audit source' (with 'Server audit' and 'Database audit' options) and a checkbox for 'Show only audit records for SQL injections'.

Clicking **View dashboard** at the top of the **Audit records** page will open a dashboard displaying audit logs info, where you can drill down into Security Insights, Access to Sensitive Data and more. This dashboard is designed to help you gain security insights for your data. You can also customize the time range and search query.

The screenshot shows the 'Audit records' page again. The top navigation bar and buttons are identical to the previous screenshot. A note below the buttons says: 'Click here to learn more about alternative methods for viewing & analyzing audit records.' Underneath, there are sections for 'Audit source' (with 'Server audit' and 'Database audit' options) and a checkbox for 'Show only audit records for SQL injections'.



- Alternatively, you can also access the audit logs from Log Analytics blade. Open your Log Analytics workspace and under **General** section, click **Logs**. You can start with a simple query, such as: `search "SQLSecurityAuditEvents"` to view the audit logs. From here, you can also use [Azure Monitor logs](#) to run advanced searches on your audit log data. Azure Monitor logs gives you real-time operational insights using integrated search and custom dashboards to readily analyze millions of records across all your workloads and servers. For additional useful information about Azure Monitor logs search language and commands, see [Azure Monitor logs search reference](#).

If you chose to write audit logs to Event Hub:

- To consume audit logs data from Event Hub, you will need to set up a stream to consume events and write them to a target. For more information, see [Azure Event Hubs Documentation](#).
- Audit logs in Event Hub are captured in the body of [Apache Avro](#) events and stored using JSON formatting with UTF-8 encoding. To read the audit logs, you can use [Avro Tools](#) or similar tools that process this format.

If you chose to write audit logs to an Azure storage account, there are several methods you can use to view the logs:

NOTE

Auditing on [Read-Only Replicas](#) is automatically enabled. For further details about the hierarchy of the storage folders, naming conventions, and log format, see the [SQL Database Audit Log Format](#).

- Audit logs are aggregated in the account you chose during setup. You can explore audit logs by using a tool such as [Azure Storage Explorer](#). In Azure storage, auditing logs are saved as a collection of blob files within a container named **sqlbauditlogs**. For further details about the hierarchy of the storage folders, naming conventions, and log format, see the [SQL Database Audit Log Format](#).
- Use the [Azure portal](#). Open the relevant database. At the top of the database's **Auditing** page, click **View audit logs**.

The screenshot shows the Azure portal interface for managing a database. The title bar says 'Alon_demo_db - Auditing'. The left sidebar has three options: 'Advanced Threat Protection', 'Auditing' (which is selected and highlighted in blue), and 'Dynamic Data Masking'. The main content area has a large 'i' icon and text stating 'If Blob Auditing is enabled on the server, it will always be a database setting.' Below this, it says 'Server-level Auditing: Disabled'. At the top right of the main area, there are 'Save', 'Discard', 'View audit logs' (with a red box around it), and 'Feedback' buttons.

Audit records opens, from which you'll be able to view the logs.

- You can view specific dates by clicking **Filter** at the top of the **Audit records** page.
- You can switch between audit records that were created by the *server audit policy* and the *database audit policy* by toggling **Audit Source**.
- You can view only SQL injection related audit records by checking **Show only audit records for SQL injections** checkbox.

The screenshot shows the 'Audit records' page for the 'Alon_demo_db' database. At the top, there's a 'Refresh' button and a 'Filter' button (highlighted with a red box). Below that, an 'i' icon provides information on viewing and analyzing audit records. The 'Audit source' dropdown is set to 'Server audit' (highlighted with a red box). There's also a checked checkbox for 'Show only audit records for SQL injections' (highlighted with a red box).

- Use the system function **sys.fn_get_audit_file** (T-SQL) to return the audit log data in tabular format. For more information on using this function, see [sys.fn_get_audit_file](#).
- Use **Merge Audit Files** in SQL Server Management Studio (starting with SSMS 17):

1. From the SSMS menu, select **File > Open > Merge Audit Files**.

The screenshot shows the 'File' menu in SSMS. The 'Open' option is highlighted with a yellow box. In the dropdown menu, 'Merge Audit Files...' is also highlighted with a yellow box. Other options in the dropdown include 'Project/Solution...', 'Merge Extended Event Files...', 'File...', and 'File...'. The 'File' menu itself has options like 'File', 'Edit', 'View', 'Debug', 'Tools', 'Window', and 'Help'.

2. The **Add Audit Files** dialog box opens. Select one of the **Add** options to choose whether to merge audit files from a local disk or import them from Azure Storage. You are required to provide your Azure Storage details and account key.
3. After all files to merge have been added, click **OK** to complete the merge operation.
4. The merged file opens in SSMS, where you can view and analyze it, as well as export it to an XEL or CSV file, or to a table.

- Use Power BI. You can view and analyze audit log data in Power BI. For more information and to access a downloadable template, see [Analyze audit log data in Power BI](#).
- Download log files from your Azure Storage blob container via the portal or by using a tool such as [Azure Storage Explorer](#).
 - After you have downloaded a log file locally, double-click the file to open, view, and analyze the logs in SSMS.
 - You can also download multiple files simultaneously via Azure Storage Explorer. To do so, right-click a specific subfolder and select **Save as** to save in a local folder.
- Additional methods:
 - After downloading several files or a subfolder that contains log files, you can merge them locally as described in the SSMS Merge Audit Files instructions described previously.
 - View blob auditing logs programmatically:
 - [Query Extended Events Files](#) by using PowerShell.

Production practices

With geo-replicated databases, when you enable auditing on the primary database the secondary database will have an identical auditing policy. It is also possible to set up auditing on the secondary database by enabling auditing on the **secondary server**, independently from the primary database.

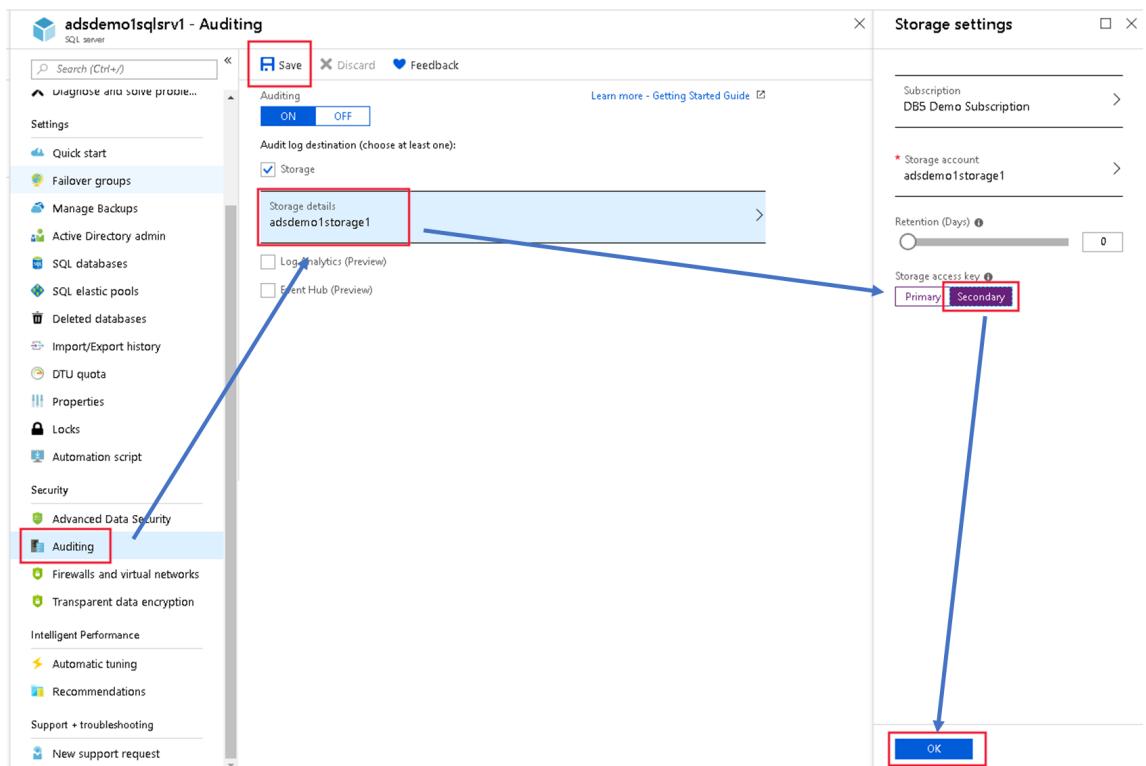
- Server-level (**recommended**): Turn on auditing on both the **primary server** as well as the **secondary server** - the primary and secondary databases will each be audited independently based on their respective server-level policy.
- Database-level: Database-level auditing for secondary databases can only be configured from Primary database auditing settings.
 - Auditing must be enabled on the *primary database itself*, not the server.
 - After auditing is enabled on the primary database, it will also become enabled on the secondary database.

IMPORTANT

With database-level auditing, the storage settings for the secondary database will be identical to those of the primary database, causing cross-regional traffic. We recommend that you enable only server-level auditing, and leave the database-level auditing disabled for all databases.

In production, you are likely to refresh your storage keys periodically. When writing audit logs to Azure storage, you need to resave your auditing policy when refreshing your keys. The process is as follows:

1. Open **Storage Details**. In the **Storage Access Key** box, select **Secondary**, and click **OK**. Then click **Save** at the top of the auditing configuration page.



2. Go to the storage configuration page and regenerate the primary access key.

3. Go back to the auditing configuration page, switch the storage access key from secondary to primary, and then click **OK**. Then click **Save** at the top of the auditing configuration page.
4. Go back to the storage configuration page and regenerate the secondary access key (in preparation for the next key's refresh cycle).

Additional Information

- If you want to customize the audited events, you can do this via [PowerShell cmdlets](#) or the [REST API](#).
- After you've configured your auditing settings, you can turn on the new threat detection feature and configure emails to receive security alerts. When you use threat detection, you receive proactive alerts

on anomalous database activities that can indicate potential security threats. For more information, see [Getting started with threat detection](#).

- For details about the log format, hierarchy of the storage folder and naming conventions, see the [Blob Audit Log Format Reference](#).

IMPORTANT

Azure SQL Database Audit stores 4000 characters of data for character fields in an audit record. When the **statement** or the **data_sensitivity_information** values returned from an auditable action contain more than 4000 characters, any data beyond the first 4000 characters will be **truncated and not audited**.

- Audit logs are written to **Append Blobs** in an Azure Blob storage on your Azure subscription:
 - Premium Storage** is currently **not supported** by Append Blobs.
- The default auditing policy includes all actions and the following set of action groups, which will audit all the queries and stored procedures executed against the database, as well as successful and failed logins:

BATCH_COMPLETED_GROUP
SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP
FAILED_DATABASE_AUTHENTICATION_GROUP

You can configure auditing for different types of actions and action groups using PowerShell, as described in the [Manage SQL database auditing using Azure PowerShell](#) section.

- When using AAD Authentication, failed logins records will *not* appear in the SQL audit log. To view failed login audit records, you need to visit the [Azure Active Directory portal](#), which logs details of these events.
- Azure SQL Database auditing is optimized for availability & performance. During very high activity Azure SQL Database allows operations to proceed and may not record some audited events.
- For configuring Immutable Auditing on storage account, see [Allow protected append blobs writes](#). Please note that the container name for Auditing is **sqlauditlogs**.

IMPORTANT

The allow protected append blobs writes setting under time-based retention is currently available and visible only in the following regions:

- East US
- South Central US
- West US 2

Manage Azure SQL Server and Database auditing using Azure PowerShell

PowerShell cmdlets (including WHERE clause support for additional filtering):

- Create or Update Database Auditing Policy ([Set-AzSqlDatabaseAudit](#))
- Create or Update Server Auditing Policy ([Set-AzSqlServerAudit](#))
- Get Database Auditing Policy ([Get-AzSqlDatabaseAudit](#))
- Get Server Auditing Policy ([Get-AzSqlServerAudit](#))
- Remove Database Auditing Policy ([Remove-AzSqlDatabaseAudit](#))

- Remove Server Auditing Policy ([Remove-AzSqlServerAudit](#))

For a script example, see [Configure auditing and threat detection using PowerShell](#).

Manage Azure SQL Server and Database auditing using REST API

REST API:

- Create or Update Database Auditing Policy
- Create or Update Server Auditing Policy
- Get Database Auditing Policy
- Get Server Auditing Policy

Extended policy with WHERE clause support for additional filtering:

- Create or Update Database *Extended* Auditing Policy
- Create or Update Server *Extended* Auditing Policy
- Get Database *Extended* Auditing Policy
- Get Server *Extended* Auditing Policy

Manage Azure SQL Server and Database auditing using Azure Resource Manager templates

You can manage Azure SQL database auditing using [Azure Resource Manager](#) templates, as shown in these examples:

- Deploy an Azure SQL Server with Auditing enabled to write audit logs to Azure Blob storage account
- Deploy an Azure SQL Server with Auditing enabled to write audit logs to Log Analytics
- Deploy an Azure SQL Server with Auditing enabled to write audit logs to Event Hubs

NOTE

The linked samples are on an external public repository and are provided 'as is', without warranty, and are not supported under any Microsoft support program/service.

Advanced data security for Azure SQL Database

1/21/2020 • 3 minutes to read • [Edit Online](#)

Advanced data security is a unified package for advanced SQL security capabilities. It includes functionality for discovering and classifying sensitive data, surfacing and mitigating potential database vulnerabilities, and detecting anomalous activities that could indicate a threat to your database. It provides a single go-to location for enabling and managing these capabilities.

Overview

Advanced data security (ADS) provides a set of advanced SQL security capabilities, including data discovery & classification, vulnerability assessment, and Advanced Threat Protection.

- [Data discovery & classification](#) provides capabilities built into Azure SQL Database for discovering, classifying, labeling & protecting the sensitive data in your databases. It can be used to provide visibility into your database classification state, and to track the access to sensitive data within the database and beyond its borders.
- [Vulnerability assessment](#) is an easy to configure service that can discover, track, and help you remediate potential database vulnerabilities. It provides visibility into your security state, and includes actionable steps to resolve security issues, and enhance your database fortifications.
- [Advanced Threat Protection](#) detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit your database. It continuously monitors your database for suspicious activities, and provides immediate security alerts on potential vulnerabilities, SQL injection attacks, and anomalous database access patterns. Advanced Threat Protection alerts provide details of the suspicious activity and recommend action on how to investigate and mitigate the threat.

Enable SQL ADS once to enable all of these included features. With one click, you can enable ADS for all databases on your SQL Database server or managed instance. Enabling or managing ADS settings requires belonging to the [SQL security manager](#) role, SQL database admin role or SQL server admin role.

ADS pricing aligns with Azure Security Center standard tier, where each protected SQL Database server or managed instance is counted as one node. Newly protected resources qualify for a free trial of Security Center standard tier. For more information, see the [Azure Security Center pricing page](#).

Getting Started with ADS

The following steps get you started with ADS.

1. Enable ADS

Enable ADS by navigating to **Advanced Data Security** under the **Security** heading for your SQL Database server or managed instance. To enable ADS for all databases on the database server or managed instance, click **Enable Advanced Data Security on the server**.

NOTE

A storage account is automatically created and configured to store your **Vulnerability Assessment** scan results. If you've already enabled ADS for another server in the same resource group and region, then the existing storage account is used.

The screenshot shows the 'samplecrm demo - Advanced Data Security' blade in the Azure portal. On the left, there's a sidebar with various navigation options like Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Settings, Security, and Audit. The 'Security' section is expanded, and 'Advanced Data Security' is selected, which is highlighted with a red box. At the top, there's a message about enabling Advanced Data Security for all databases on the server. Below that, a large blue button says 'Enable Advanced Data Security on the server'. To the right, there are three main cards:

- Data Discovery & Classification (preview)**: Shows 0 TOTAL recommended columns to classify. Below it, a table for 'Recommended columns to classify' has two columns: COLUMN and SENSITIVITY LABEL.
- Vulnerability Assessment**: Shows 0 TOTAL failed checks. Below it, a table for 'Failed Checks' has two columns: SECURITY CHECK and RISK. A legend indicates: HIGH RISK FAILURES (dark red), MEDIUM RISK FAILURES (medium red), and LOW RISK FAILURES (light red).
- Advanced Threat Protection**: Shows 0 TOTAL security alerts. Below it, a table for 'Security Alerts' has two columns: DESCRIPTION and DATE. A legend indicates: HIGH SEVERITY ALERTS (dark blue) and MEDIUM SEVERITY ALERTS (medium blue).

NOTE

The cost of ADS is aligned with Azure Security Center standard tier pricing per node, where a node is the entire SQL Database server or managed instance. You are thus paying only once for protecting all databases on the database server or managed instance with ADS. You can try ADS out initially with a free trial.

2. Start classifying data, tracking vulnerabilities, and investigating threat alerts

Click the **Data Discovery & Classification** card to see recommended sensitive columns to classify and to classify your data with persistent sensitivity labels. Click the **Vulnerability Assessment** card to view and manage vulnerability scans and reports, and to track your security stature. If security alerts have been received, click the **Advanced Threat Protection** card to view details of the alerts and to see a consolidated report on all alerts in your Azure subscription via the Azure Security Center security alerts page.

3. Manage ADS settings on your SQL Database server or managed instance

To view and manage ADS settings, navigate to **Advanced Data Security** under the **Security** heading for your SQL Database server or managed instance. On this page, you can enable or disable ADS, and modify vulnerability assessment and Advanced Threat Protection settings for your entire SQL Database server or managed instance.

crm-demo - Advanced Data Security

Save Discard Feedback

ADVANCED DATA SECURITY

ON OFF

Advanced Data Security costs 15 USD/server/month. It includes Data Discovery & Classification, Vulnerability Assessment and Advanced Threat Protection. We invite you to a trial period for the first 30 days, without charge.

VULNERABILITY ASSESSMENT SETTINGS

Subscription >

Storage account vastore >

Periodic recurring scans ON OFF

Send scan reports to @microsoft.com ✓

Email service and co-administrators

ADVANCED THREAT PROTECTION SETTINGS

Send alerts to @microsoft.com

Email service and co-administrators

Advanced Threat Protection types All >

Advanced Data Security

Auditing

4. Manage ADS settings for a SQL database

To override ADS settings for a particular database, check the **Enable Advanced Data Security at the database level** checkbox. Use this option only if you have a particular requirement to receive separate Advanced Threat Protection alerts or vulnerability assessment results for the individual database, in place of or in addition to the alerts and results received for all databases on the database server or managed instance.

Once the checkbox is selected, you can then configure the relevant settings for this database.

Database settings

samplecrmdemo

Save Discard Feedback

Advanced Data Security for this database: **Enabled**

[View Advanced Data Security server settings](#)

Enable Advanced Data Security at the database level i

VULNERABILITY ASSESSMENT SETTINGS

Subscription >

Storage account >

Periodic recurring scans i

Send scan reports to i

Email service and co-administrators

ADVANCED THREAT PROTECTION SETTINGS

Send alerts to i

✓

Email service and co-administrators

Advanced Threat Protection types >

All

⚠ Storage details i

samplecrmdemo >

Advanced data security settings for your database server or managed instance can also be reached from the ADS database pane. Click **Settings** in the main ADS pane, and then click **View Advanced Data Security server settings**.

Database settings

samplecrmdemo

Save Discard Feedback

Advanced Data Security for this database: **Enabled**

[View Advanced Data Security server settings](#)

Enable Advanced Data Security at the database level i

Next steps

- Learn more about [data discovery & classification](#)
- Learn more about [vulnerability assessment](#)
- Learn more about [Advanced Threat Protection](#)
- Learn more about [Azure security center](#)

Azure SQL Database and SQL Data Warehouse data discovery & classification

1/23/2020 • 6 minutes to read • [Edit Online](#)

Data discovery & classification provides advanced capabilities built into Azure SQL Database for **discovering, classifying, labeling & reporting** the sensitive data in your databases.

Discovering and classifying your most sensitive data (business, financial, healthcare, personally identifiable data (PII), and so on.) can play a pivotal role in your organizational information protection stature. It can serve as infrastructure for:

- Helping meet data privacy standards and regulatory compliance requirements.
- Various security scenarios, such as monitoring (auditing) and alerting on anomalous access to sensitive data.
- Controlling access to and hardening the security of databases containing highly sensitive data.

Data discovery & classification is part of the [Advanced Data Security](#) (ADS) offering, which is a unified package for advanced SQL security capabilities. Data discovery & classification can be accessed and managed via the central SQL ADS portal.

NOTE

This document relates to Azure SQL Database and Azure SQL Data Warehouse. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse. For SQL Server (on premises), see [SQL Data Discovery and Classification](#).

What is data discovery & classification

Data discovery & classification introduces a set of advanced services and new SQL capabilities, forming a new SQL Information Protection paradigm aimed at protecting the data, not just the database:

- **Discovery & recommendations**

The classification engine scans your database and identifies columns containing potentially sensitive data. It then provides you an easy way to review and apply the appropriate classification recommendations via the Azure portal.

- **Labeling**

Sensitivity classification labels can be persistently tagged on columns using new classification metadata attributes introduced into the SQL Engine. This metadata can then be utilized for advanced sensitivity-based auditing and protection scenarios.

- **Query result set sensitivity**

The sensitivity of query result set is calculated in real time for auditing purposes.

- **Visibility**

The database classification state can be viewed in a detailed dashboard in the portal. Additionally, you can download a report (in Excel format) to be used for compliance & auditing purposes, as well as other needs.

Discover, classify & label sensitive columns

The following section describes the steps for discovering, classifying, and labeling columns containing sensitive data in your database, as well as viewing the current classification state of your database and exporting reports.

The classification includes two metadata attributes:

- Labels – The main classification attributes, used to define the sensitivity level of the data stored in the column.
- Information Types – Provide additional granularity into the type of data stored in the column.

Define and customize your classification taxonomy

SQL data discovery & classification comes with a built-in set of sensitivity labels and a built-in set of information types and discovery logic. You now have the ability to customize this taxonomy and define a set and ranking of classification constructs specifically for your environment.

Definition and customization of your classification taxonomy is done in one central place for your entire Azure tenant. That location is in [Azure Security Center](#), as part of your Security Policy. Only someone with administrative rights on the Tenant root management group can perform this task.

As part of the Information Protection policy management, you can define custom labels, rank them, and associate them with a selected set of information types. You can also add your own custom information types and configure them with string patterns, which are added to the discovery logic for identifying this type of data in your databases. Learn more about customizing and managing your policy in the [Information Protection policy how-to guide](#).

Once the tenant-wide policy has been defined, you can continue with the classification of individual databases using your customized policy.

Classify your SQL Database

1. Go to the [Azure portal](#).
2. Navigate to **Advanced Data Security** under the Security heading in your Azure SQL Database pane. Click to enable advanced data security, and then click on the **Data discovery & classification** card.

The screenshot shows the Azure portal interface for a SQL database named 'Clinic'. The 'Advanced Threat Protection' blade is open. The 'Data Discovery & Classification' card is highlighted with a red box. It displays a summary with 0 total columns classified and a table showing three columns: Patient_PatientID (Confidential - GDPR), Email (Confidential - GDPR), and PasswordHash (Confidential). The 'Vulnerability Assessment' card shows 5 failed checks, including server-level firewall rules not granting access and sensitive data columns not being tracked. The 'Threat Detection' card shows 0 total alerts.

3. The **Overview** tab includes a summary of the current classification state of the database, including a detailed list of all classified columns, which you can also filter to view only specific schema parts, information types and labels. If you haven't yet classified any columns, [skip to step 5](#).

MayaDB - Data discovery & classification

Overview Classification

Classified columns: 15 / 109

Tables containing sensitive data: 5 / 12

Unique information types: 7

Label distribution:

- CONFIDENTIAL - GDPR: Orange
- HIGHLY CONFIDENTIAL: Red
- CONFIDENTIAL: Magenta
- GENERAL: Dark Blue
- PUBLIC: Light Blue

Information type distribution:

- NAME: Green
- BANKING: Yellow
- CONTACT INFO: Red
- CREDENTIALS: Magenta
- FINANCIAL: Dark Blue
- CREDIT CARD: Light Blue
- HEALTH: Teal

Schema	Table	Column	Information Type	Sensitivity Label
dbo	ErrorLog	UserName	Credentials	Confidential
SalesLT	Address	AddressLine1	Banking	Highly confidential
	Address	AddressLine2	Contact Info	Confidential - GDPR

- To download a report in Excel format, click on the **Export** option in the top menu of the window.

MayaDB - Data discovery & classification

Overview Classification

Export

- To begin classifying your data, click on the **Classification tab** at the top of the window.

Classification

Classified columns

Tables containing s

- The classification engine scans your database for columns containing potentially sensitive data and provides a list of **recommended column classifications**. To view and apply classification recommendations:

- To view the list of recommended column classifications, click on the recommendations panel at the bottom of the window:

15 columns with classification recommendations

- Review the list of recommendations – to accept a recommendation for a specific column, check the checkbox in the left column of the relevant row. You can also mark *all recommendations* as accepted by checking the checkbox in the recommendations table header.

The screenshot shows the MayaDB - Data discovery & classification interface. The left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), SETTINGS, Pricing tier (scale DTUs), Geo-Replication, Auditing & Threat Detection, Vulnerability Assessment (Pr...), Data discovery & classificatio..., Dynamic Data Masking, Transparent data encryption, Connection strings, and Sync to other databases. The main area has tabs for Overview and Classification, with Classification selected. A search bar at the top right includes Save, Discard, Add classification, and Feedback buttons. Below the search bar, there are filters for Schema (0 selected), Table (0 selected), Filter by column, Information type (0 sele...), and Sensitivity label (0 sele...). The main content area displays a table titled '15 columns with classification recommendations'. The table has columns: SCHEMA, TABLE, COLUMN, INFORMATION TYPE, and SENSITIVITY LABEL. Several rows are listed, each with a checkbox in the first column. A red box highlights the 'Accept selected recommendations' button at the bottom of the table.

- To apply the selected recommendations, click on the blue **Accept selected recommendations** button.

This is a screenshot of a modal dialog box titled '15 columns with classification recommendations'. It contains a single button labeled 'Accept selected recommendations' which is highlighted with a red border.

- You can also **manually classify** columns as an alternative, or in addition, to the recommendation-based classification:

- Click on **Add classification** in the top menu of the window.

The screenshot shows the MayaDB - Data discovery & classification interface. The left sidebar includes links for Overview and Activity log. The main area has tabs for Overview and Classification, with Classification selected. A search bar at the top right includes Save, Discard, Add classification, and Feedback buttons. The 'Add classification' button is highlighted with a red box.

- In the context window that opens, select the schema > table > column that you want to classify, and the information type and sensitivity label. Then click on the blue **Add classification** button at the bottom of the context window.

The screenshot shows the 'Add classification' dialog box overlaid on the main 'MayaDB - Data discovery & classification' page. The dialog box has several input fields: 'Schema name' set to 'SalesLT', 'Table name' set to 'Customer', 'Column name' set to 'EmailAddress (nvarchar)', 'Information type' set to 'Contact Info', and 'Sensitivity label' set to 'Confidential - GDPR'. The 'Add classification' button at the bottom right is highlighted with a red box.

- To complete your classification and persistently label (tag) the database columns with the new classification metadata, click on **Save** in the top menu of the window.

The screenshot shows the main 'MayaDB - Data discovery & classification' page. The top menu bar includes 'Save' (highlighted with a red box), 'Discard', 'Add classification', and 'Feedback'. Below the menu, there are tabs for 'Overview' and 'Classification'.

Auditing access to sensitive data

An important aspect of the information protection paradigm is the ability to monitor access to sensitive data. [Azure SQL Database Auditing](#) has been enhanced to include a new field in the audit log called *data_sensitivity_information*, which logs the sensitivity classifications (labels) of the actual data that was returned by the query.

d	client_ip	application_name	duration_milliseconds	response_rows	affected_rows	connection_id	data_sensitivity_information
	7.125	Microsoft SQL Server Management Studio - Query	1	847	847	C244A066-2271...	Confidential - GDPR
	7.125	Microsoft SQL Server Management Studio - Query	2	32	32	C244A066-2271...	Confidential
	7.125	Microsoft SQL Server Management Studio - Query	41	32	32	A7088FD4-759E...	Confidential, Confidential - GDPR

Permissions

The following built-in roles can read the data classification of an Azure SQL database: [Owner](#), [Reader](#), [Contributor](#), [SQL Security Manager](#) and [User Access Administrator](#).

The following built-in roles can modify the data classification of an Azure SQL database: [Owner](#), [Contributor](#), [SQL Security Manager](#).

Learn more about [RBAC for Azure resources](#)

Manage classifications

- [T-SQL](#)

- [Rest APIs](#)
- [PowerShell Cmdlet](#)

You can use T-SQL to add/remove column classifications, as well as retrieve all classifications for the entire database.

NOTE

When using T-SQL to manage labels, there is no validation that labels added to a column exist in the organizational information protection policy (the set of labels that appear in the portal recommendations). It is therefore up to you to validate this.

- Add/update the classification of one or more columns: [ADD SENSITIVITY CLASSIFICATION](#)
- Remove the classification from one or more columns: [DROP SENSITIVITY CLASSIFICATION](#)
- View all classifications on the database: [sys.sensitivity_classifications](#)

Next steps

- Learn more about [advanced data security](#).
- Consider configuring [Azure SQL Database Auditing](#) for monitoring and auditing access to your classified sensitive data.
- For a YouTube presentation that includes Data Discovery & Classification, see [Discovering, classifying, labeling & protecting SQL data | Data Exposed](#).

SQL Vulnerability Assessment service helps you identify database vulnerabilities

2/19/2020 • 6 minutes to read • [Edit Online](#)

SQL Vulnerability Assessment is an easy to configure service that can discover, track, and help you remediate potential database vulnerabilities. Use it to proactively improve your database security.

Vulnerability Assessment is part of the [advanced data security](#) (ADS) offering, which is a unified package for advanced SQL security capabilities. Vulnerability Assessment can be accessed and managed via the central SQL ADS portal.

NOTE

Vulnerability Assessment is supported for Azure SQL Database, Azure SQL Managed Instance and Azure SQL Data Warehouse. For simplicity, SQL Database is used in this article when referring to any of these managed database services.

The Vulnerability Assessment service

SQL Vulnerability Assessment (VA) is a service that provides visibility into your security state, and includes actionable steps to resolve security issues, and enhance your database security. It can help you:

- Meet compliance requirements that require database scan reports.
- Meet data privacy standards.
- Monitor a dynamic database environment where changes are difficult to track.

Vulnerability Assessment is a scanning service built into the Azure SQL Database service. The service employs a knowledge base of rules that flag security vulnerabilities and highlight deviations from best practices, such as misconfigurations, excessive permissions, and unprotected sensitive data. The rules are based on Microsoft's best practices and focus on the security issues that present the biggest risks to your database and its valuable data. They cover both database-level issues as well as server-level security issues, like server firewall settings and server-level permissions. These rules also represent many of the requirements from various regulatory bodies to meet their compliance standards.

Results of the scan include actionable steps to resolve each issue and provide customized remediation scripts where applicable. An assessment report can be customized for your environment by setting an acceptable baseline for permission configurations, feature configurations, and database settings.

Implementing Vulnerability Assessment

The following steps implement VA on SQL Database.

1. Run a scan

Get started with VA by navigating to **Advanced Data Security** under the Security heading in your Azure SQL Database pane. Click to enable advanced data security, and then click on **Select Storage** or on the **Vulnerability Assessment** card, which automatically opens the Vulnerability Assessment settings card for the entire SQL server.

Start by configuring a storage account where your scan results for all databases on the server will be stored. For information about storage accounts, see [About Azure storage accounts](#). Once storage is configured, click **Scan** to scan your database for vulnerabilities.

Home > Clinic - Advanced Threat Protection > Vulnerability Assessment

Vulnerability Assessment

[Scan](#) [Export Scan Results](#) [Scan History](#) [Settings](#) [Feedback](#)

Total failing checks 0	Total passing checks 0	Risk summary	Last scan time	Learn more	
		High Risk 0			
		Medium Risk 0			
		Low Risk 0			

Failed (0) Passed (0) Click 'Scan' to scan your database for vulnerabilities.

Filter by ID or security check Category: All selected Risk: All selected

ID	SECURITY CHECK	APPLIES TO	CATEGORY	RISK	ADDITIONAL INFO
No results					

NOTE

The scan is lightweight and safe. It takes a few seconds to run, and is entirely read-only. It does not make any changes to your database.

2. View the report

When your scan is complete, your scan report is automatically displayed in the Azure portal. The report presents an overview of your security state: how many issues were found and their respective severities. Results include warnings on deviations from best practices and a snapshot of your security-related settings, such as database principals and roles and their associated permissions. The scan report also provides a map of sensitive data discovered in your database, and includes recommendations to classify that data using [data discovery & classification](#).

Home > ContosoCRMDB - Advanced Threat Protection > Vulnerability Assessment

Vulnerability Assessment

[Scan](#) [Export Scan Results](#) [Scan History](#) [Settings](#) [Feedback](#)

Total failing checks 6	Total passing checks 42	Risk summary	Last scan time	Learn more	
		High Risk 2			
		Medium Risk 3			
		Low Risk 1			

Failed (6) Passed (42)

Filter by ID or security check Category: All selected Risk: All selected

ID	SECURITY CHECK	APPLIES TO	CATEGORY	RISK	ADDITIONAL INFO
VA2108	Minimal set of principals should be members of fixed high impact database roles	ContosoCRMDB	Authentication & Auth...	High	No baseline set
VA20...	Server-level firewall rules should be tracked and maintained at a strict minimum	master	Surface area reduction	High	No baseline set
VA10...	Excessive permissions should not be granted to PUBLIC role	ContosoCRMDB	Authentication & Auth...	Medium	
VA1281	All memberships for user-defined roles should be intended	ContosoCRMDB	Auditing & Logging	Medium	No baseline set
VA1288	Sensitive data columns should be classified	ContosoCRMDB	Data protection	Medium	No baseline set
VA1282	Orphan roles should be removed	ContosoCRMDB	Authentication & Auth...	Low	

3. Analyze the results and resolve issues

Review your results and determine the findings in the report that are true security issues in your environment. Drill down to each failed result to understand the impact of the finding and why each security check failed. Use the actionable remediation information provided by the report to resolve the issue.

VA2108 - Minimal set of principals should be members of fixed high impact database roles

Approve As Baseline Clear Baseline

NAME	VA2108 - Minimal set of principals should be members of fixed high impact database roles				
RISK	High				
STATUS	X FAIL				
DESCRIPTION	SQL Server provides roles to help manage the permissions. Roles are security principals that group other principals. Database-level roles are database-wide in their permission scope. This rule checks that a minimal set of principals are members of the fixed database roles.				
IMPACT	Fixed database roles may have administrative permissions on the system. Following the principle of least privilege, it is important to minimize membership in fixed database roles and keep a baseline of these memberships. See https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles for additional information on database roles.				
RULE QUERY	<pre>SELECT user_name(sr.member_principal_id) as [Principal], user_name(sr.role_principal_id) as [Role], type_desc as [Principal Type]</pre> Run in Query Editor				
MICROSOFT RECOMMENDATION	Empty Set				
RESULTS	IN BASELINE	PRINCIPAL	ROLE	PRINCIPAL TYPE	AUTHENTICATION TYPE
	X	michael8	db_ddladmin	SQL_USER	NONE
	X	test1	db_ddladmin	DATABASE_ROLE	NONE

REMEDIALION

Remove members who should not have access to the database role

REMEDIALION SCRIPT

```
ALTER ROLE [db_ddladmin] DROP MEMBER [michael8]  
ALTER ROLE [db_ddladmin] DROP MEMBER [test1]
```

[Run in Query Editor](#)

4. Set your baseline

As you review your assessment results, you can mark specific results as being an acceptable *Baseline* in your environment. The baseline is essentially a customization of how the results are reported. Results that match the baseline are considered as passing in subsequent scans. Once you have established your baseline security state, VA only reports on deviations from the baseline and you can focus your attention on the relevant issues.

VA2108 - Minimal set of principals should be members of fixed high impact database roles

Approve As Baseline Clear Baseline

NAME	VA2108 - Minimal set of principals should be members of fixed high impact database roles																		
RISK	High																		
STATUS	✖ FAIL																		
DESCRIPTION	SQL Server provides roles to help manage the permissions. Roles are security principals that group other principals. Database-level roles are database-wide in their permission scope. This rule checks that a minimal set of principals are members of the fixed database roles.																		
IMPACT	Fixed database roles may have administrative permissions on the system. Following the principle of least privilege, it is important to minimize membership in fixed database roles and keep a baseline of these memberships. See https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles for additional information on database roles.																		
RULE QUERY	<pre>SELECT user_name(sr.member_principal_id) as [Principal], user_name(sr.role_principal_id) as [Role], type_desc as [Principal]</pre> Run in Query Editor																		
MICROSOFT RECOMMENDATION	Empty Set																		
RESULTS	<table border="1"> <thead> <tr> <th>IN BASELINE</th> <th>PRINCIPAL</th> <th>ROLE</th> <th>PRINCIPAL TYPE</th> <th>AUTHENTICATION TYPE</th> </tr> </thead> <tbody> <tr> <td>✖</td> <td>michael8</td> <td>db_ddladmin</td> <td>SQL_USER</td> <td>NONE</td> </tr> <tr> <td>✖</td> <td>test1</td> <td>db_ddladmin</td> <td>DATABASE_ROLE</td> <td>NONE</td> </tr> </tbody> </table>				IN BASELINE	PRINCIPAL	ROLE	PRINCIPAL TYPE	AUTHENTICATION TYPE	✖	michael8	db_ddladmin	SQL_USER	NONE	✖	test1	db_ddladmin	DATABASE_ROLE	NONE
IN BASELINE	PRINCIPAL	ROLE	PRINCIPAL TYPE	AUTHENTICATION TYPE															
✖	michael8	db_ddladmin	SQL_USER	NONE															
✖	test1	db_ddladmin	DATABASE_ROLE	NONE															

5. Run a new scan to see your customized tracking report

After you complete setting up your **Rule Baselines**, run a new scan to view the customized report. VA now reports only the security issues that deviate from your approved baseline state.

Home > ContosoCRMDB - Advanced Threat Protection > Vulnerability Assessment

Vulnerability Assessment

✖ Scan ✔ Export Scan Results ⌚ Scan History ⚙️ Settings ❤️ Feedback

Total failing checks	Total passing checks	Risk summary	Last scan time	Learn more
3 ✖	45 ✔	High Risk: 0 Medium Risk: 2 Low Risk: 1	Thu, 17 May 2018 09:50:22 UTC	SQL Security Center Best Practices for SQL Security

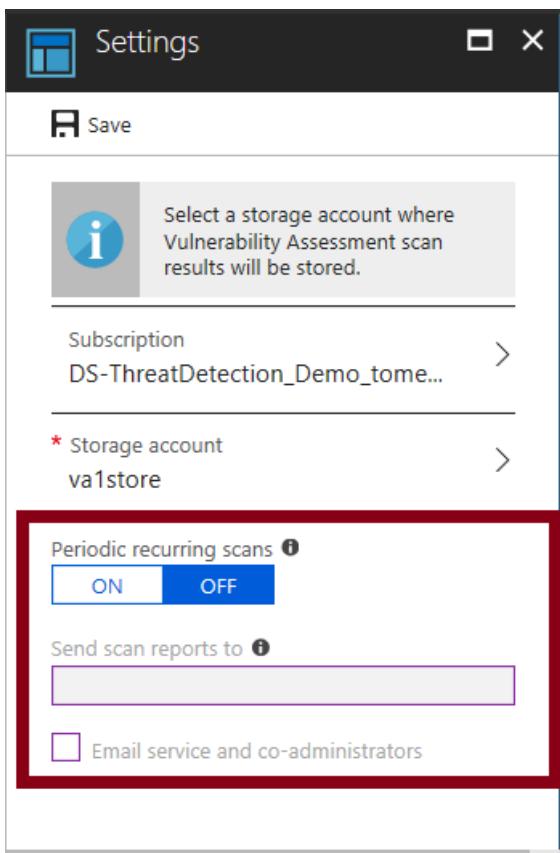
✖ Failed (3) ✔ Passed (45)

ID	SECURITY CHECK	APPLIES TO	CATEGORY	STATUS
VA1281	All memberships for user-defined roles should be intended	ContosoCRMDB	Auditing & Logging	✔ PASS (per custom baseline)
VA2108	Minimal set of principals should be members of fixed high impact database roles	ContosoCRMDB	Authentication & Auth...	✔ PASS (per custom baseline)
VA20...	Server-level firewall rules should be tracked and maintained at a strict minimum	master	Surface area reduction	✔ PASS (per custom baseline)
VA1020	Server principal GUEST should not be a member of any role	ContosoCRMDB	Authentication & Auth...	✔ PASS
VA20...	Database-level firewall rules should not grant excessive access	ContosoCRMDB	Surface area reduction	✔ PASS
VA20...	Database-level firewall rules should be tracked and maintained at a strict minimum	ContosoCRMDB	Surface area reduction	✔ PASS
VA10...	Excessive permissions should not be granted to PUBLIC role on objects or columns	ContosoCRMDB	Authentication & Auth...	✔ PASS
VA10...	Principal GUEST should not be granted permissions in the database	ContosoCRMDB	Authentication & Auth...	✔ PASS

Vulnerability Assessment can now be used to monitor that your database maintains a high level of security at all times, and that your organizational policies are met. If compliance reports are required, VA reports can be helpful to facilitate the compliance process.

6. Set up periodic recurring scans

Navigate to the Vulnerability Assessment settings to turn on **Periodic recurring scans**. This configures Vulnerability Assessment to automatically run a scan on your database once per week. A scan result summary will be sent to the email address(es) you provide.



7. Export an assessment report

Click **Export Scan Results** to create a downloadable Excel report of your scan result. This report contains a summary tab that displays a summary of the assessment, including all failed checks. It also includes a **Results** tab containing the full set of results from the scan, including all checks that were run and the result details for each.

8. View scan history

Click **Scan History** in the VA pane to view a history of all scans previously run on this database. Select a particular scan in the list to view the detailed results of that scan.

Vulnerability Assessment can now be used to monitor that your database maintains a high level of security at all times, and that your organizational policies are met. If compliance reports are required, VA reports can be helpful to facilitate the compliance process.

Manage Vulnerability Assessments using Azure PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

You can use Azure PowerShell cmdlets to programmatically manage your vulnerability assessments. The supported cmdlets are:

- [Update-AzSqlDatabaseVulnerabilityAssessmentSetting](#)

Updates the vulnerability assessment settings of a database

- [Get-AzSqlDatabaseVulnerabilityAssessmentSetting](#)

Returns the vulnerability assessment settings of a database

- [Clear-AzSqlDatabaseVulnerabilityAssessmentSetting](#)

Clears the vulnerability assessment settings of a database

- [Set-AzSqlDatabaseVulnerabilityAssessmentRuleBaseline](#)

Sets the vulnerability assessment rule baseline.

- [Get-AzSqlDatabaseVulnerabilityAssessmentRuleBaseline](#)

Gets the vulnerability assessment rule baseline for a given rule.

- [Clear-AzSqlDatabaseVulnerabilityAssessmentRuleBaseline](#)

Clears the vulnerability assessment rule baseline. First set the baseline before using this cmdlet to clear it.

- [Start-AzSqlDatabaseVulnerabilityAssessmentScan](#)

Triggers the start of a vulnerability assessment scan

- [Get-AzSqlDatabaseVulnerabilityAssessmentScanRecord](#)

Gets all vulnerability assessment scan record(s) associated with a given database.

- [Convert-AzSqlDatabaseVulnerabilityAssessmentScan](#)

Converts vulnerability assessment scan results to an Excel file

For a script example, see [Azure SQL Vulnerability Assessment PowerShell support](#).

Manage Vulnerability Assessments baseline rules using Resource Manager templates

To configure Vulnerability Assessment baselines by using Azure Resource Manager templates, use the

`Microsoft.Sql/servers/databases/vulnerabilityAssessments/rules/baselines` type.

Ensure that you have enabled vulnerabilityAssements on the computer that's running SQL Server before you add baselines.

Here's an example for defining Baseline Rule VA2065 to masterdb and VA1143 userdb as resources in a Resource Manager template:

```

"resources": [
    {
        "type": "Microsoft.Sql/servers/databases/vulnerabilityAssessments/rules/baselines",
        "apiVersion": "2018-06-01-preview",
        "name": "[concat(parameters('server_name'), '/', parameters('database_name') ,
'/default/VA2065/master')]",
        "properties": {
            "baselineResults": [
                {
                    "result": [
                        "FirewallRuleName3",
                        "StartIpAddress",
                        "EndIpAddress"
                    ]
                },
                {
                    "result": [
                        "FirewallRuleName4",
                        "62.92.15.68",
                        "62.92.15.68"
                    ]
                }
            ]
        },
        {
            "type": "Microsoft.Sql/servers/databases/vulnerabilityAssessments/rules/baselines",
            "apiVersion": "2018-06-01-preview",
            "name": "[concat(parameters('server_name'), '/', parameters('database_name'),
'/default/VA2130/Default')]",
            "dependsOn": [
                "[resourceId('Microsoft.Sql/servers/vulnerabilityAssessments', parameters('server_name'),
'Default')]"
            ],
            "properties": {
                "baselineResults": [
                    {
                        "result": [
                            "dbo"
                        ]
                    }
                ]
            }
        }
    ]
}

```

For user database and master database, the resource names are defined differently:

- Master database - "name": "[concat(parameters('server_name'), '/', parameters('database_name') ,
'/default/VA2065/**master**')]",
- User database - "name": "[concat(parameters('server_name'), '/', parameters('database_name') ,
'/default/VA2065/**default**')]",

To handle **Boolean** types as true/false, set the baseline result with binary input like "1"/"0".

```
{  
    "type": "Microsoft.Sql/servers/databases/vulnerabilityAssessments/rules/baselines",  
    "apiVersion": "2018-06-01-preview",  
    "name": "[concat(parameters('server_name'), '/', parameters('database_name'),  
'/default/VA1143/Default')]",  
    "dependsOn": [  
        "[resourceId('Microsoft.Sql/servers/vulnerabilityAssessments', parameters('server_name'),  
'Default')]"  
    ],  
    "properties": {  
        "baselineResults": [  
            {  
                "result": [  
                    "1"  
                ]  
            }  
        ]  
    }  
}
```

Next steps

- Learn more about [advanced data security](#)
- Learn more about [data discovery & classification](#)

Advanced Threat Protection for Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

Advanced Threat Protection for [Azure SQL Database](#) and [SQL Data Warehouse](#) detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases.

Advanced Threat Protection is part of the [Advanced data security](#) (ADS) offering, which is a unified package for advanced SQL security capabilities. Advanced Threat Protection can be accessed and managed via the central SQL ADS portal.

NOTE

This topic applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

What is Advanced Threat Protection

Advanced Threat Protection provides a new layer of security, which enables customers to detect and respond to potential threats as they occur by providing security alerts on anomalous activities. Users receive an alert upon suspicious database activities, potential vulnerabilities, and SQL injection attacks, as well as anomalous database access and queries patterns. Advanced Threat Protection integrates alerts with [Azure Security Center](#), which include details of suspicious activity and recommend action on how to investigate and mitigate the threat.

Advanced Threat Protection makes it simple to address potential threats to the database without the need to be a security expert or manage advanced security monitoring systems.

For a full investigation experience, it is recommended to enable [SQL Database Auditing](#), which writes database events to an audit log in your Azure storage account.

Advanced Threat Protection alerts

Advanced Threat Protection for Azure SQL Database detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases and it can trigger the following alerts:

- **Vulnerability to SQL injection:** This alert is triggered when an application generates a faulty SQL statement in the database. This alert may indicate a possible vulnerability to SQL injection attacks. There are two possible reasons for the generation of a faulty statement:
 - A defect in application code that constructs the faulty SQL statement
 - Application code or stored procedures don't sanitize user input when constructing the faulty SQL statement, which may be exploited for SQL Injection
- **Potential SQL injection:** This alert is triggered when an active exploit happens against an identified application vulnerability to SQL injection. This means the attacker is trying to inject malicious SQL statements using the vulnerable application code or stored procedures.
- **Access from unusual location:** This alert is triggered when there is a change in the access pattern to SQL server, where someone has logged on to the SQL server from an unusual geographical location. In some cases, the alert detects a legitimate action (a new application or developer maintenance). In other cases, the alert detects a malicious action (former employee, external attacker).

- **Access from unusual Azure data center:** This alert is triggered when there is a change in the access pattern to SQL server, where someone has logged on to the SQL server from an unusual Azure data center that was seen on this server during the recent period. In some cases, the alert detects a legitimate action (your new application in Azure, Power BI, Azure SQL Query Editor). In other cases, the alert detects a malicious action from an Azure resource/service (former employee, external attacker).
- **Access from unfamiliar principal:** This alert is triggered when there is a change in the access pattern to SQL server, where someone has logged on to the SQL server using an unusual principal (SQL user). In some cases, the alert detects a legitimate action (new application, developer maintenance). In other cases, the alert detects a malicious action (former employee, external attacker).
- **Access from a potentially harmful application:** This alert is triggered when a potentially harmful application is used to access the database. In some cases, the alert detects penetration testing in action. In other cases, the alert detects an attack using common attack tools.
- **Brute force SQL credentials:** This alert is triggered when there is an abnormal high number of failed logins with different credentials. In some cases, the alert detects penetration testing in action. In other cases, the alert detects brute force attack.

Explore anomalous database activities upon detection of a suspicious event

You receive an email notification upon detection of anomalous database activities. The email provides information on the suspicious security event including the nature of the anomalous activities, database name, server name, application name, and the event time. In addition, the email provides information on possible causes and recommended actions to investigate and mitigate the potential threat to the database.

Azure SQL database



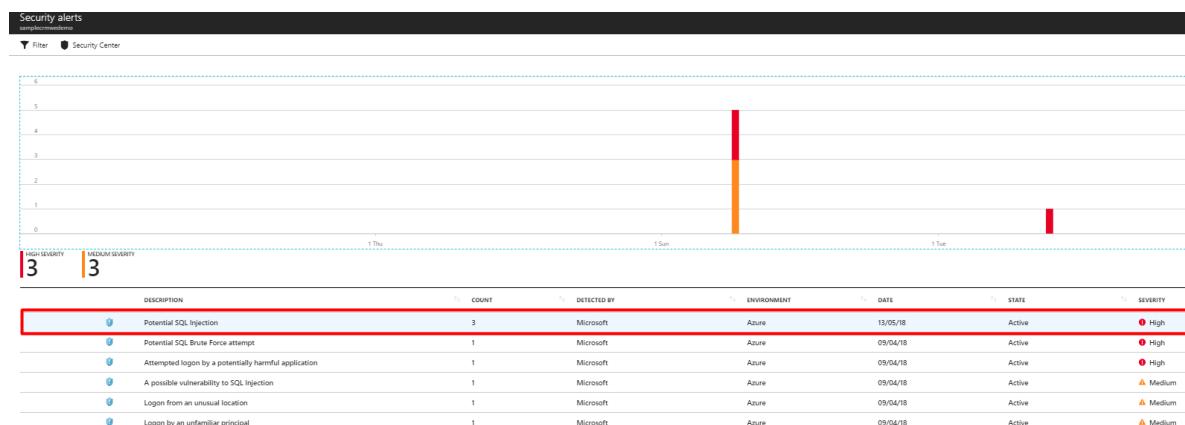
Potential exploitation of application code vulnerability to SQL Injection was detected. This may indicate a SQL Injection attack on database 'samplecrmwedemo'.

[View recent SQL alerts](#)

Activity details

Severity	High
Subscription ID	
Subscription Name	DS-THREATDETECTION_DEMO_TOMERR_R&D_60843
Server	
Database	
IP address	
Principal Name	de*****
Application	.Net SqlClient Data Provider
Date	May 13, 2018 12:09:12 UTC
Threat ID	1
Potential causes	Defect in application code constructing SQL statements; application code doesn't sanitize user input and was exploited to inject malicious SQL statements.
Investigation steps	View the vulnerable SQL statement
Remediation steps	Read more about SQL Injection threat and how to fix the vulnerable application code.

- Click the **View recent SQL alerts** link in the email to launch the Azure portal and show the Azure Security Center alerts page, which provides an overview of active threats detected on the SQL database.



- Click a specific alert to get additional details and actions for investigating this threat and remediating future threats.

For example, SQL injection is one of the most common Web application security issues on the Internet that is used to attack data-driven applications. Attackers take advantage of application vulnerabilities to

inject malicious SQL statements into application entry fields, breaching or modifying data in the database. For SQL Injection alerts, the alert's details include the vulnerable SQL statement that was exploited.

Potential SQL Injection
samplecrmwedemo

[Learn more](#)

General information

DESCRIPTION	Potential SQL Injection was detected on your database samplecrmwedemo on server ronmatwedemo
DETECTION TIME	Sunday, 13 May 2018, 3:09:12 pm
SEVERITY	! High
STATE	Active
ATTACKED RESOURCE	samplecrmwedemo
SUBSCRIPTION	
DETECTED BY	■■■ Microsoft
ACTION TAKEN	Detected
ENVIRONMENT	Azure
RESOURCE TYPE	SQL Server
SERVER	
DATABASE	
IP ADDRESS	
PRINCIPAL NAME	dev1
APPLICATION	.Net SqlClient Data Provider
VULNERABLE STATEMENT	SELECT * FROM sqli_users WHERE username = ''OR 1 = 1 --' AND password = 'dfdfdfdf'
THREAT ID	1

Remediation steps

INVESTIGATION STEPS	View the vulnerable SQL statement
REMEDIATION STEPS	Read more about SQL Injection threat and how to fix the vulnerable application code.

Explore Advanced Threat Protection alerts for your database in the Azure portal

Advanced Threat Protection integrates its alerts with [Azure security center](#). Live SQL Advanced Threat Protection tiles within the database and SQL ADS blades in the Azure portal track the status of active threats.

Click **Advanced Threat Protection alert** to launch the Azure Security Center alerts page and get an overview of active SQL threats detected on the database or data warehouse.

samplecrmeusdemo (ronmateusdemo/samplecrmeusdemo)

Search (Ctrl+ /)

Copy Restore Export Set server firewall Delete Connect with... Feedback

Overview Activity log Tags Diagnose and solve problems Quick start Query editor (preview)

Settings Configure Geo-Replication Connection strings Sync to other databases Add Azure Search Properties Locks Automation script

Security Advanced Data Security Auditing

Resource utilization (samplecrmeusdemo)

1 hour 24 hour

DTU percentage (Max) ronmateusdemo/samplecrmeusdemo 0 %

Database data storage

Used space 7 MB Allocated space 16 MB Max size 250 GB

0% USED SPACE

Notifications (5) Database features (6)

All Alerts (5) Recommendations (0) Info (0)

Advanced Threat Protection alerts

There are 5 alerts (2 critical) for this database. Click here to review.

samplecrmeusdemo (ronmateusdemo/samplecrmeusdemo) - Advanced Data Security

Search (Ctrl+ /)

Settings Feedback

Overview Activity log Tags Diagnose and solve problems Quick start Query editor (preview)

Settings Configure Geo-Replication Connection strings Sync to other databases Add Azure Search Properties Locks Automation script

Security Advanced Data Security Auditing

Data Discovery & Classification (preview)

10 TOTAL

CONFIDENTIAL CONFIDENTIAL - GDPR

Recommended columns to classify

COLUMN	SENSITIVITY LABEL
There are no active recommendations at the moment.	

Vulnerability Assessment

1 TOTAL

HIGH RISK FAILURES MEDIUM RISK FAILURES LOW RISK FAILURES

Failed Checks

SECURITY CHECK	RISK
Sensitive data columns should be classified	Medium

Advanced Threat Protection

5 TOTAL

HIGH SEVERITY ALERTS MEDIUM SEVERITY ALERTS

Security Alerts

DESCRIPTION	DATE
Potential SQL Injection	2/20/2019
Potential SQL Brute Force attempt	12/16/2018
Logon from an unusual location	12/16/2018
...	

Next steps

- Learn more about [Advanced Threat Protection in single and pooled databases](#).
- Learn more about [Advanced Threat Protection in managed instance](#).
- Learn more about [Advanced data security](#).
- Learn more about [Azure SQL Database auditing](#)
- Learn more about [Azure security center](#)
- For more information on pricing, see the [SQL Database pricing page](#)

2 minutes to read

Azure SQL Database and Data Warehouse network access controls

2/14/2020 • 6 minutes to read • [Edit Online](#)

NOTE

This article applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

IMPORTANT

This article does *not* apply to **Azure SQL Database Managed Instance**. for more information about the networking configuration, see [connecting to a Managed Instance](#).

When you create a new Azure SQL Server [from Azure portal](#), the result is a public endpoint in the format `yourservername.database.windows.net`. By design, all access to the public endpoint is denied. You can then use the following network access controls to selectively allow access to the SQL Database via the public endpoint

- Allow Azure Services: - When set to ON, other resources within the Azure boundary, for example an Azure Virtual Machine, can access SQL Database
- IP firewall rules: - Use this feature to explicitly allow connections from a specific IP address, for example from on-premises machines.
- Virtual Network firewall rules: - Use this feature to allow traffic from a specific Virtual Network within the Azure boundary

Allow Azure services

During creation of a new Azure SQL Server [from Azure portal](#), this setting is left unchecked.

New server X

Microsoft

* Server name
mysqlsrvr ✓
.database.windows.net

* Server admin login
azureuser ✓

* Password
***** ✓

* Confirm password
***** ✓

* Location
(US) West US

Allow Azure services to access server i

You can also change this setting via the firewall pane after the Azure SQL Server is created as follows.

mysqlserver - Firewalls and virtual networks

SQL server

Search (Ctrl+ /)

Save Discard Add client IP

Overview

Activity log

Access control (IAM)

Tags

i Connections from the IPs specified below

Allow access to Azure services

ON OFF

When set to **ON** Azure SQL Server allows communications from all resources inside the Azure boundary, that may or may not be part of your subscription.

In many cases, the **ON** setting is more permissive than what most customers want. They may want to set this setting to **OFF** and replace it with more restrictive IP firewall rules or Virtual Network firewall rules. Doing so affects the following features that run on VMs in Azure that are not part of your VNet and hence connect to SQL Database via an Azure IP address.

Import Export Service

Import Export Service does not work **Allow Azure services to access server** set to OFF. However you can work around the problem [by manually running sqlpackage.exe from an Azure VM or performing the export](#) directly in your code by using the DACFx API.

Data Sync

To use the Data sync feature with **Allow Azure services to access server** set to OFF, you need to create individual firewall rule entries to [add IP addresses](#) from the **Sql service tag** for the region hosting the **Hub** database. Add these server level firewall rules to the logical servers hosting both **Hub** and **Member** databases (which may be in different regions)

Use the following PowerShell script to generate the IP addresses corresponding to SQL service tag for West US region

```
PS C:\> $serviceTags = Get-AzNetworkServiceTag -Location eastus2
PS C:\> $sql = $serviceTags.Values | Where-Object { $_.Name -eq "Sql.WestUS" }
PS C:\> $sql.Properties.AddressPrefixes.Count
70
PS C:\> $sql.Properties.AddressPrefixes
13.86.216.0/25
13.86.216.128/26
13.86.216.192/27
13.86.217.0/25
13.86.217.128/26
13.86.217.192/27
```

TIP

Get-AzNetworkServiceTag returns the global range for Sql Service Tag despite specifying the Location parameter. Be sure to filter it to the region that hosts the Hub database used by your sync group

Note that the output of the PowerShell script is in Classless Inter-Domain Routing (CIDR) notation and this needs to be converted to a format of Start and End IP address using [Get-IPRangeStartEnd.ps1](#) like this

```
PS C:\> Get-IPRangeStartEnd -ip 52.229.17.93 -cidr 26
start      end
----      ---
52.229.17.64 52.229.17.127
```

Do the following additional steps to convert all the IP addresses from CIDR to Start and End IP address format.

```
PS C:\>foreach( $i in $sql.Properties.AddressPrefixes) {$ip,$cidr= $i.split('/') ; Get-IPRangeStartEnd -ip $ip
-cidr $cidr;}
start      end
----      ---
13.86.216.0    13.86.216.127
13.86.216.128  13.86.216.191
13.86.216.192  13.86.216.223
```

You can now add these as distinct firewall rules and then set **Allow Azure services to access server** to OFF.

IP firewall rules

Ip based firewall is a feature of Azure SQL Server that prevents all access to your database server until you explicitly [add IP addresses](#) of the client machines.

Virtual Network firewall rules

In addition to IP rules, the Azure SQL Server firewall allows you to define *virtual network rules*.

To learn more, see [Virtual Network service endpoints and rules for Azure SQL Database](#) or watch this video:

Azure Networking terminology

Be aware of the following Azure Networking terms as you explore Virtual Network firewall rules

Virtual network: You can have virtual networks associated with your Azure subscription

Subnet: A virtual network contains **subnets**. Any Azure virtual machines (VMs) that you have are assigned to subnets. One subnet can contain multiple VMs or other compute nodes. Compute nodes that are outside of your virtual network cannot access your virtual network unless you configure your security to allow access.

Virtual Network service endpoint: A [Virtual Network service endpoint](#) is a subnet whose property values include one or more formal Azure service type names. In this article we are interested in the type name of **Microsoft.Sql**, which refers to the Azure service named SQL Database.

Virtual network rule: A virtual network rule for your SQL Database server is a subnet that is listed in the access control list (ACL) of your SQL Database server. To be in the ACL for your SQL Database, the subnet must contain the **Microsoft.Sql** type name. A virtual network rule tells your SQL Database server to accept communications from every node that is on the subnet.

IP vs. Virtual Network firewall rules

The Azure SQL Server firewall allows you to specify IP address ranges from which communications are accepted into SQL Database. This approach is fine for stable IP addresses that are outside the Azure private network. However, virtual machines (VMs) within the Azure private network are configured with *dynamic* IP addresses. Dynamic IP addresses can change when your VM is restarted and in turn invalidate the IP-based firewall rule. It would be folly to specify a dynamic IP address in a firewall rule, in a production environment.

You can work around this limitation by obtaining a *static* IP address for your VM. For details, see [Configure private IP addresses for a virtual machine by using the Azure portal](#). However, the static IP approach can become difficult to manage, and it is costly when done at scale.

Virtual network rules are easier alternative to establish and to manage access from a specific subnet that contains your VMs.

NOTE

You cannot yet have SQL Database on a subnet. If your Azure SQL Database server was a node on a subnet in your virtual network, all nodes within the virtual network could communicate with your SQL Database. In this case, your VMs could communicate with SQL Database without needing any virtual network rules or IP rules.

Next steps

- For a quickstart on creating a server-level IP firewall rule, see [Create an Azure SQL database](#).
- For a quickstart on creating a server-level Vnet firewall rule, see [Virtual Network service endpoints and rules for Azure SQL Database](#).
- For help with connecting to an Azure SQL database from open source or third-party applications, see [Client quickstart code samples to SQL Database](#).
- For information on additional ports that you may need to open, see the **SQL Database: Outside vs inside** section of [Ports beyond 1433 for ADO.NET 4.5 and SQL Database](#)
- For an overview of Azure SQL Database Connectivity, see [Azure SQL Connectivity Architecture](#)
- For an overview of Azure SQL Database security, see [Securing your database](#)

Azure SQL Database and Azure SQL Data Warehouse IP firewall rules

12/20/2019 • 12 minutes to read • [Edit Online](#)

NOTE

This article applies to Azure SQL servers, and to both Azure SQL Database and Azure SQL Data Warehouse databases on an Azure SQL server. For simplicity, *SQL Database* is used to refer to both SQL Database and SQL Data Warehouse.

IMPORTANT

This article does *not* apply to *Azure SQL Database Managed Instance*. For information about network configuration, see [Connect your application to Azure SQL Database Managed Instance](#).

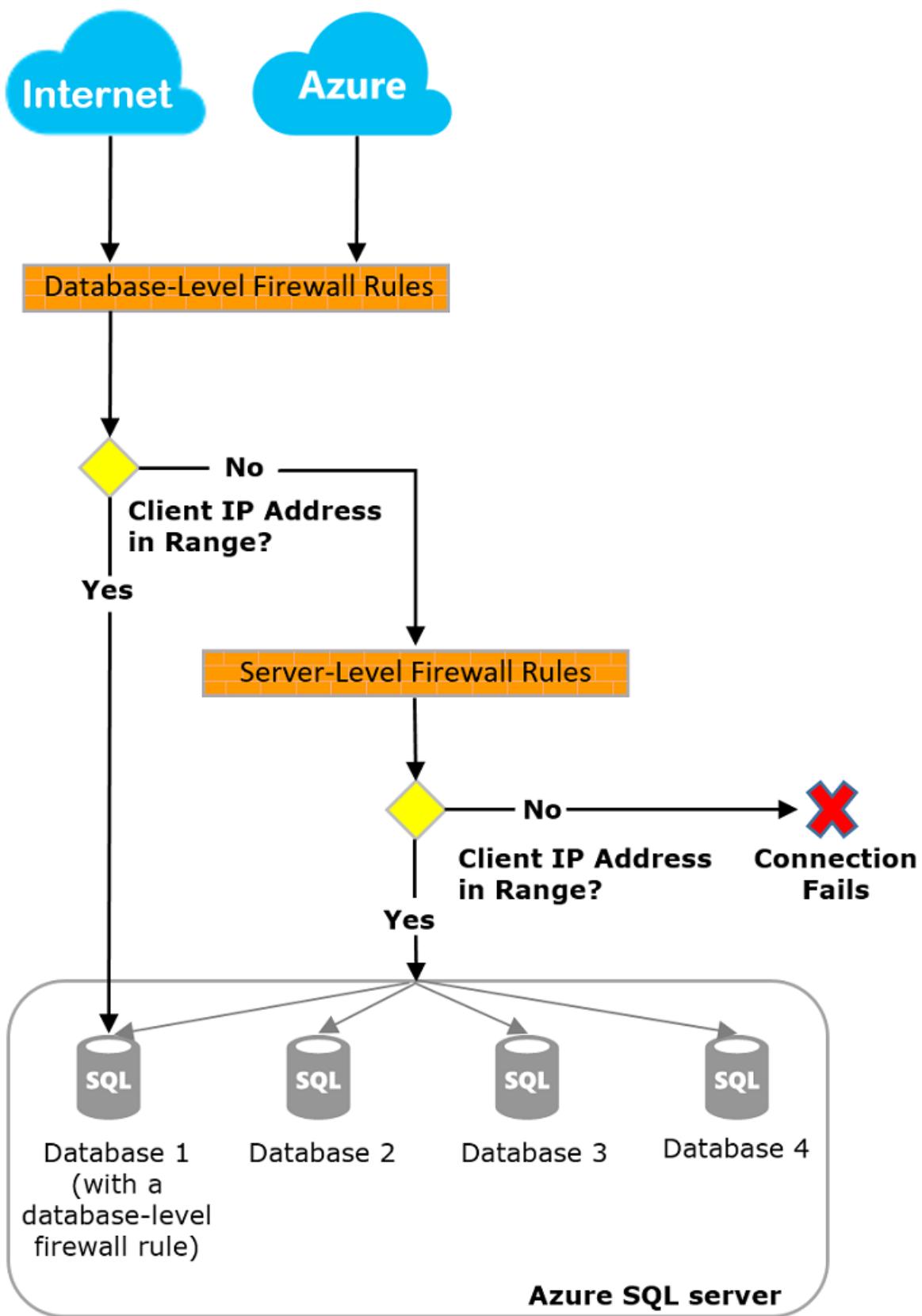
When you create a new Azure SQL server named *mysqlserver*, for example, the SQL Database firewall blocks all access to the public endpoint for the server (which is accessible at *mysqlserver.database.windows.net*).

IMPORTANT

SQL Data Warehouse only supports server-level IP firewall rules. It doesn't support database-level IP firewall rules.

How the firewall works

Connection attempts from the internet and Azure must pass through the firewall before they reach your SQL server or SQL database, as the following diagram shows.



Server-level IP firewall rules

These rules enable clients to access your entire Azure SQL server, that is, all the databases within the same SQL Database server. The rules are stored in the *master* database. You can have a maximum of 128 server-level IP firewall rules for an Azure SQL Server.

You can configure server-level IP firewall rules by using the Azure portal, PowerShell, or Transact-SQL statements.

- To use the portal or PowerShell, you must be the subscription owner or a subscription contributor.
- To use Transact-SQL, you must connect to the SQL Database instance as the server-level principal login or

as the Azure Active Directory administrator. (A server-level IP firewall rule must first be created by a user who has Azure-level permissions.)

Database-level IP firewall rules

These rules enable clients to access certain (secure) databases within the same SQL Database server. You create the rules for each database (including the *master* database), and they're stored in the individual database.

You can only create and manage database-level IP firewall rules for master and user databases by using Transact-SQL statements and only after you configure the first server-level firewall.

If you specify an IP address range in the database-level IP firewall rule that's outside the range in the server-level IP firewall rule, only those clients that have IP addresses in the database-level range can access the database.

You can have a maximum of 128 database-level IP firewall rules for a database. For more information about configuring database-level IP firewall rules, see the example later in this article and see [sp_set_database_firewall_rule \(Azure SQL Database\)](#).

Recommendations for how to set firewall rules

We recommend that you use database-level IP firewall rules whenever possible. This practice enhances security and makes your database more portable. Use server-level IP firewall rules for administrators. Also use them when you have many databases that have the same access requirements, and you don't want to configure each database individually.

NOTE

For information about portable databases in the context of business continuity, see [Authentication requirements for disaster recovery](#).

Server-level versus database-level IP firewall rules

Should users of one database be fully isolated from another database?

If yes, use database-level IP firewall rules to grant access. This method avoids using server-level IP firewall rules, which permit access through the firewall to all databases. That would reduce the depth of your defenses.

Do users at the IP addresses need access to all databases?

If yes, use server-level IP firewall rules to reduce the number of times that you have to configure IP firewall rules.

Does the person or team who configures the IP firewall rules only have access through the Azure portal, PowerShell, or the REST API?

If so, you must use server-level IP firewall rules. Database-level IP firewall rules can only be configured through Transact-SQL.

Is the person or team who configures the IP firewall rules prohibited from having high-level permission at the database level?

If so, use server-level IP firewall rules. You need at least *CONTROL DATABASE* permission at the database level to configure database-level IP firewall rules through Transact-SQL.

Does the person or team who configures or audits the IP firewall rules centrally manage IP firewall rules for many (perhaps hundreds) of databases?

In this scenario, best practices are determined by your needs and environment. Server-level IP firewall rules

might be easier to configure, but scripting can configure rules at the database-level. And even if you use server-level IP firewall rules, you might need to audit database-level IP firewall rules to see if users with **CONTROL** permission on the database create database-level IP firewall rules.

Can I use a mix of server-level and database-level IP firewall rules?

Yes. Some users, such as administrators, might need server-level IP firewall rules. Other users, such as users of a database application, might need database-level IP firewall rules.

Connections from the internet

When a computer tries to connect to your database server from the internet, the firewall first checks the originating IP address of the request against the database-level IP firewall rules for the database that the connection requests.

- If the address is within a range that's specified in the database-level IP firewall rules, the connection is granted to the SQL database that contains the rule.
- If the address isn't within a range in the database-level IP firewall rules, the firewall checks the server-level IP firewall rules. If the address is within a range that's in the server-level IP firewall rules, the connection is granted. Server-level IP firewall rules apply to all SQL databases on the Azure SQL server.
- If the address isn't within a range that's in any of the database-level or server-level IP firewall rules, the connection request fails.

NOTE

To access SQL Database from your local computer, ensure that the firewall on your network and local computer allow outgoing communication on TCP port 1433.

Connections from inside Azure

To allow applications hosted inside Azure to connect to your SQL server, Azure connections must be enabled. When an application from Azure tries to connect to your database server, the firewall verifies that Azure connections are allowed. A firewall setting that has starting and ending IP addresses equal to **0.0.0.0** indicates that Azure connections are allowed. This can be turned on directly from the Azure Portal blade by setting Firewall rules, as well as switching the **Allow Azure Services and resources to access this server** to **ON** in the **Firewalls and virtual networks** settings. If the connection isn't allowed, the request doesn't reach the SQL Database server.

IMPORTANT

This option configures the firewall to allow all connections from Azure, including connections from the subscriptions of other customers. If you select this option, make sure that your login and user permissions limit access to authorized users only.

Create and manage IP firewall rules

You create the first server-level firewall setting by using the [Azure portal](#) or programmatically by using [Azure PowerShell](#), [Azure CLI](#), or an [Azure REST API](#). You create and manage additional server-level IP firewall rules by using these methods or Transact-SQL.

IMPORTANT

Database-level IP firewall rules can only be created and managed by using Transact-SQL.

To improve performance, server-level IP firewall rules are temporarily cached at the database level. To refresh

the cache, see [DBCC FLUSHAUTHCACHE](#).

TIP

You can use [SQL Database Auditing](#) to audit server-level and database-level firewall changes.

Use the Azure portal to manage server-level IP firewall rules

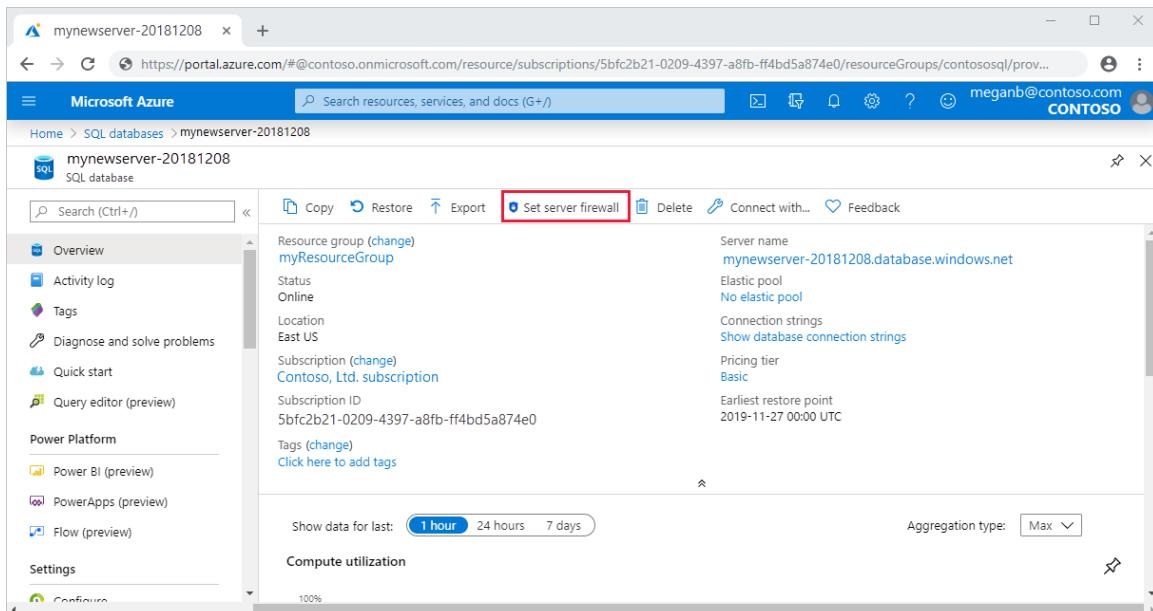
To set a server-level IP firewall rule in the Azure portal, go to the overview page for your Azure SQL database or your SQL Database server.

TIP

For a tutorial, see [Create a DB using the Azure portal](#).

From the database overview page

1. To set a server-level IP firewall rule from the database overview page, select **Set server firewall** on the toolbar, as the following image shows.



The **Firewall settings** page for the SQL Database server opens.

2. Select **Add client IP** on the toolbar to add the IP address of the computer that you're using, and then select **Save**. A server-level IP firewall rule is created for your current IP address.

From the server overview page

The overview page for your server opens. It shows the fully qualified server name (such as `mynewserver20170403.database.windows.net`) and provides options for further configuration.

1. To set a server-level rule from this page, select **Firewall** from the **Settings** menu on the left side.
2. Select **Add client IP** on the toolbar to add the IP address of the computer that you're using, and then select **Save**. A server-level IP firewall rule is created for your current IP address.

Use Transact-SQL to manage IP firewall rules

CATALOG VIEW OR STORED PROCEDURE	LEVEL	DESCRIPTION
<code>sys.firewall_rules</code>	Server	Displays the current server-level IP firewall rules
<code>sp_set_firewall_rule</code>	Server	Creates or updates server-level IP firewall rules
<code>sp_delete_firewall_rule</code>	Server	Removes server-level IP firewall rules
<code>sys.database_firewall_rules</code>	Database	Displays the current database-level IP firewall rules
<code>sp_set_database_firewall_rule</code>	Database	Creates or updates the database-level IP firewall rules
<code>sp_delete_database_firewall_rule</code>	Databases	Removes database-level IP firewall rules

The following example reviews the existing rules, enables a range of IP addresses on the server *Contoso*, and deletes an IP firewall rule:

```
SELECT * FROM sys.firewall_rules ORDER BY name;
```

Next, add a server-level IP firewall rule.

```
EXECUTE sp_set_firewall_rule @name = N'ContosoFirewallRule',
    @start_ip_address = '192.168.1.1', @end_ip_address = '192.168.1.10'
```

To delete a server-level IP firewall rule, execute the `sp_delete_firewall_rule` stored procedure. The following example deletes the rule `ContosoFirewallRule`:

```
EXECUTE sp_delete_firewall_rule @name = N'ContosoFirewallRule'
```

Use PowerShell to manage server-level IP firewall rules

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all development is now for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az and AzureRm modules are substantially identical.

CMDLET	LEVEL	DESCRIPTION
Get-AzSqlServerFirewallRule	Server	Returns the current server-level firewall rules
New-AzSqlServerFirewallRule	Server	Creates a new server-level firewall rule
Set-AzSqlServerFirewallRule	Server	Updates the properties of an existing server-level firewall rule
Remove-AzSqlServerFirewallRule	Server	Removes server-level firewall rules

The following example uses PowerShell to set a server-level IP firewall rule:

```
New-AzSqlServerFirewallRule -ResourceGroupName "myResourceGroup" `  
    -ServerName $servername `  
    -FirewallRuleName "ContosoIPRange" -StartIpAddress "192.168.1.0" -EndIpAddress "192.168.1.255"
```

TIP

For \$servername specify the server name and not the fully qualified DNS name e.g. specify `mysqlDbserver` instead of `mysqlDbserver.database.windows.net`

TIP

For PowerShell examples in the context of a quickstart, see [Create DB - PowerShell](#) and [Create a single database and configure a SQL Database server-level IP firewall rule using PowerShell](#).

Use CLI to manage server-level IP firewall rules

CMDLET	LEVEL	DESCRIPTION
az sql server firewall-rule create	Server	Creates a server IP firewall rule
az sql server firewall-rule list	Server	Lists the IP firewall rules on a server
az sql server firewall-rule show	Server	Shows the detail of an IP firewall rule
az sql server firewall-rule update	Server	Updates an IP firewall rule
az sql server firewall-rule delete	Server	Deletes an IP firewall rule

The following example uses CLI to set a server-level IP firewall rule:

```
az sql server firewall-rule create --resource-group myResourceGroup --server $servername \
-n ContosoIPRange --start-ip-address 192.168.1.0 --end-ip-address 192.168.1.255
```

TIP

For \$servername specify the server name and not the fully qualified DNS name e.g. specify **mysqldbserver** instead of **mysqldbserver.database.windows.net**

TIP

For a CLI example in the context of a quickstart, see [Create DB - Azure CLI](#) and [Create a single database and configure a SQL Database IP firewall rule using the Azure CLI](#).

Use a REST API to manage server-level IP firewall rules

API	LEVEL	DESCRIPTION
List firewall rules	Server	Displays the current server-level IP firewall rules
Create or update firewall rules	Server	Creates or updates server-level IP firewall rules
Delete firewall rules	Server	Removes server-level IP firewall rules
Get firewall rules	Server	Gets server-level IP firewall rules

Troubleshoot the database firewall

Consider the following points when access to the SQL Database service doesn't behave as you expect.

- **Local firewall configuration:**

Before your computer can access SQL Database, you may need to create a firewall exception on your computer for TCP port 1433. To make connections inside the Azure cloud boundary, you may have to open additional ports. For more information, see the "SQL Database: Outside vs inside" section of [Ports beyond 1433 for ADO.NET 4.5 and SQL Database](#).

- **Network address translation:**

Because of network address translation (NAT), the IP address that's used by your computer to connect to SQL Database may be different than the IP address in your computer's IP configuration settings. To view the IP address that your computer is using to connect to Azure:

1. Sign in to the portal.
2. Go to the **Configure** tab on the server that hosts your database.
3. The **Current Client IP Address** is displayed in the **Allowed IP Addresses** section. Select **Add** for **Allowed IP Addresses** to allow this computer to access the server.

- **Changes to the allow list haven't taken effect yet:**

There may be up to a five-minute delay for changes to the SQL Database firewall configuration to take effect.

- **The login isn't authorized, or an incorrect password was used:**

If a login doesn't have permissions on the SQL Database server or the password is incorrect, the connection to the server is denied. Creating a firewall setting only gives clients an *opportunity* to try to connect to your server. The client must still provide the necessary security credentials. For more information about preparing logins, see [Controlling and granting database access to SQL Database and SQL Data Warehouse](#).

- **Dynamic IP address:**

If you have an internet connection that uses dynamic IP addressing and you have trouble getting through the firewall, try one of the following solutions:

- Ask your internet service provider for the IP address range that's assigned to your client computers that access the SQL Database server. Add that IP address range as an IP firewall rule.
- Get static IP addressing instead for your client computers. Add the IP addresses as IP firewall rules.

Next steps

- Confirm that your corporate network environment allows inbound communication from the compute IP address ranges (including SQL ranges) that are used by the Azure datacenters. You might have to add those IP addresses to the allow list. See [Microsoft Azure datacenter IP ranges](#).
- For a quickstart about creating a server-level IP firewall rule, see [Create an Azure SQL database](#).
- For help with connecting to an Azure SQL database from open-source or third-party applications, see [Client quickstart code samples to SQL Database](#).
- For information about additional ports that you may need to open, see the "SQL Database: Outside vs inside" section of [Ports beyond 1433 for ADO.NET 4.5 and SQL Database](#)
- For an overview of Azure SQL Database security, see [Securing your database](#).

Use virtual network service endpoints and rules for database servers

1/14/2020 • 12 minutes to read • [Edit Online](#)

Virtual network rules are one firewall security feature that controls whether the database server for your single databases and elastic pool in Azure [SQL Database](#) or for your databases in [SQL Data Warehouse](#) accepts communications that are sent from particular subnets in virtual networks. This article explains why the virtual network rule feature is sometimes your best option for securely allowing communication to your Azure SQL Database and SQL Data Warehouse.

IMPORTANT

This article applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse. This article does *not* apply to a **managed instance** deployment in Azure SQL Database because it does not have a service endpoint associated with it.

To create a virtual network rule, there must first be a [virtual network service endpoint](#) for the rule to reference.

How to create a virtual network rule

If you only create a virtual network rule, you can skip ahead to the steps and explanation [later in this article](#).

Details about virtual network rules

This section describes several details about virtual network rules.

Only one geographic region

Each Virtual Network service endpoint applies to only one Azure region. The endpoint does not enable other regions to accept communication from the subnet.

Any virtual network rule is limited to the region that its underlying endpoint applies to.

Server-level, not database-level

Each virtual network rule applies to your whole Azure SQL Database server, not just to one particular database on the server. In other words, virtual network rule applies at the server-level, not at the database-level.

- In contrast, IP rules can apply at either level.

Security administration roles

There is a separation of security roles in the administration of Virtual Network service endpoints. Action is required from each of the following roles:

- **Network Admin:** Turn on the endpoint.
- **Database Admin:** Update the access control list (ACL) to add the given subnet to the SQL Database server.

RBAC alternative:

The roles of Network Admin and Database Admin have more capabilities than are needed to manage virtual network rules. Only a subset of their capabilities is needed.

You have the option of using [role-based access control \(RBAC\)](#) in Azure to create a single custom role that has only the necessary subset of capabilities. The custom role could be used instead of involving either the Network Admin or the Database Admin. The surface area of your security exposure is lower if you add a user to a custom role, versus adding the user to the other two major administrator roles.

NOTE

In some cases the Azure SQL Database and the VNet-subnet are in different subscriptions. In these cases you must ensure the following configurations:

- Both subscriptions must be in the same Azure Active Directory tenant.
- The user has the required permissions to initiate operations, such as enabling service endpoints and adding a VNet-subnet to the given Server.
- Both subscriptions must have the Microsoft.Sql provider registered.

Limitations

For Azure SQL Database, the virtual network rules feature has the following limitations:

- In the firewall for your SQL Database, each virtual network rule references a subnet. All these referenced subnets must be hosted in the same geographic region that hosts the SQL Database.
- Each Azure SQL Database server can have up to 128 ACL entries for any given virtual network.
- Virtual network rules apply only to Azure Resource Manager virtual networks; and not to [classic deployment model](#) networks.
- Turning ON virtual network service endpoints to Azure SQL Database also enables the endpoints for the MySQL and PostgreSQL Azure services. However, with endpoints ON, attempts to connect from the endpoints to your MySQL or PostgreSQL instances may fail.
 - The underlying reason is that MySQL and PostgreSQL likely do not have a virtual network rule configured. You must configure a virtual network rule for Azure Database for MySQL and PostgreSQL and the connection will succeed.
- On the firewall, IP address ranges do apply to the following networking items, but virtual network rules do not:
 - [Site-to-Site \(S2S\) virtual private network \(VPN\)](#)
 - On-premises via [ExpressRoute](#)

Considerations when using Service Endpoints

When using service endpoints for Azure SQL Database, review the following considerations:

- **Outbound to Azure SQL Database Public IPs is required:** Network Security Groups (NSGs) must be opened to Azure SQL Database IPs to allow connectivity. You can do this by using NSG [Service Tags](#) for Azure SQL Database.

ExpressRoute

If you are using [ExpressRoute](#) from your premises, for public peering or Microsoft peering, you will need to identify the NAT IP addresses that are used. For public peering, each ExpressRoute circuit by default uses two NAT IP addresses applied to Azure service traffic when the traffic enters the Microsoft Azure network backbone. For Microsoft peering, the NAT IP address(es) that are used are either customer provided or are provided by the service provider. To allow access to your service resources, you must allow these public IP addresses in the resource IP firewall setting. To find your public peering ExpressRoute circuit IP addresses, [open a support ticket with ExpressRoute](#) via the Azure portal. Learn more about [NAT for ExpressRoute public and Microsoft peering](#).

To allow communication from your circuit to Azure SQL Database, you must create IP network rules for the public IP addresses of your NAT.

Impact of using VNet Service Endpoints with Azure storage

Azure Storage has implemented the same feature that allows you to limit connectivity to your Azure Storage account. If you choose to use this feature with an Azure Storage account that is being used by Azure SQL Server, you can run into issues. Next is a list and discussion of Azure SQL Database and Azure SQL Data Warehouse features that are impacted by this.

Azure SQL Data Warehouse PolyBase

PolyBase is commonly used to load data into Azure SQL Data Warehouse from Azure Storage accounts. If the Azure Storage account that you are loading data from limits access only to a set of VNet-subnets, connectivity from PolyBase to the Account will break. For enabling both PolyBase import and export scenarios with Azure SQL Data Warehouse connecting to Azure Storage that's secured to VNet, follow the steps indicated below:

Prerequisites

- Install Azure PowerShell using this [guide](#).
- If you have a general-purpose v1 or blob storage account, you must first upgrade to general-purpose v2 using this [guide](#).
- You must have **Allow trusted Microsoft services to access this storage account** turned on under Azure Storage account **Firewalls and Virtual networks** settings menu. Refer to this [guide](#) for more information.

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

Steps

1. In PowerShell, **register your Azure SQL Server** hosting your Azure SQL Data Warehouse instance with Azure Active Directory (AAD):

```
Connect-AzAccount  
Select-AzSubscription -SubscriptionId <subscriptionId>  
Set-AzSqlServer -ResourceGroupName your-database-server-resourceGroup -ServerName your-SQL-servername  
-AssignIdentity
```

2. Create a **general-purpose v2 Storage Account** using this [guide](#).

NOTE

- If you have a general-purpose v1 or blob storage account, you must **first upgrade to v2** using this [guide](#).
- For known issues with Azure Data Lake Storage Gen2, please refer to this [guide](#).

3. Under your storage account, navigate to **Access Control (IAM)**, and click **Add role assignment**. Assign **Storage Blob Data Contributor** RBAC role to your Azure SQL Server hosting your Azure SQL Data Warehouse which you've registered with Azure Active Directory (AAD) as in step#1.

NOTE

Only members with Owner privilege can perform this step. For various built-in roles for Azure resources, refer to this [guide](#).

4. Polybase connectivity to the Azure Storage account:

- Create a database **master key** if you haven't created one earlier:

```
CREATE MASTER KEY [ENCRYPTION BY PASSWORD = 'somepassword'];
```

- Create database scoped credential with **IDENTITY = 'Managed Service Identity'**:

```
CREATE DATABASE SCOPED CREDENTIAL msi_cred WITH IDENTITY = 'Managed Service Identity';
```

NOTE

- There is no need to specify SECRET with Azure Storage access key because this mechanism uses [Managed Identity](#) under the covers.
- IDENTITY name should be '**Managed Service Identity**' for PolyBase connectivity to work with Azure Storage account secured to VNet.

- Create external data source with `abfss://` scheme for connecting to your general-purpose v2 storage account using PolyBase:

```
CREATE EXTERNAL DATA SOURCE ext_datasource_with_abfss WITH (TYPE = hadoop, LOCATION = 'abfss://myfile@mystorageaccount.dfs.core.windows.net', CREDENTIAL = msi_cred);
```

NOTE

- If you already have external tables associated with general-purpose v1 or blob storage account, you should first drop those external tables and then drop corresponding external data source. Then create external data source with `abfss://` scheme connecting to general-purpose v2 storage account as above and re-create all the external tables using this new external data source. You could use [Generate and Publish Scripts Wizard](#) to generate create-scripts for all the external tables for ease.
- For more information on `abfss://` scheme, refer to this [guide](#).
- For more information on CREATE EXTERNAL DATA SOURCE, refer to this [guide](#).

- Query as normal using [external tables](#).

Azure SQL Database Blob Auditing

Blob auditing pushes audit logs to your own storage account. If this storage account uses the VNet Service endpoints feature then connectivity from Azure SQL Database to the storage account will break.

Adding a VNet Firewall rule to your server without turning On VNet Service Endpoints

Long ago, before this feature was enhanced, you were required to turn VNet service endpoints On before you could implement a live VNet rule in the Firewall. The endpoints related a given VNet-subnet to an Azure SQL Database. But now as of January 2018, you can circumvent this requirement by setting the

IgnoreMissingVNetServiceEndpoint flag.

Merely setting a Firewall rule does not help secure the server. You must also turn VNet service endpoints On for the security to take effect. When you turn service endpoints On, your VNet-subnet experiences downtime until it completes the transition from Off to On. This is especially true in the context of large VNets. You can use the **IgnoreMissingVNetServiceEndpoint** flag to reduce or eliminate the downtime during transition.

You can set the **IgnoreMissingVNetServiceEndpoint** flag by using PowerShell. For details, see [PowerShell to create a Virtual Network service endpoint and rule for Azure SQL Database](#).

Errors 40914 and 40615

Connection error 40914 relates to *virtual network rules*, as specified on the Firewall pane in the Azure portal. Error 40615 is similar, except it relates to *IP address rules* on the Firewall.

Error 40914

Message text: Cannot open server '*[server-name]*' requested by the login. Client is not allowed to access the server.

Error description: The client is in a subnet that has virtual network server endpoints. But the Azure SQL Database server has no virtual network rule that grants to the subnet the right to communicate with the SQL Database.

Error resolution: On the Firewall pane of the Azure portal, use the virtual network rules control to [add a virtual network rule](#) for the subnet.

Error 40615

Message text: Cannot open server '{0}' requested by the login. Client with IP address '{1}' is not allowed to access the server.

Error description: The client is trying to connect from an IP address that is not authorized to connect to the Azure SQL Database server. The server firewall has no IP address rule that allows a client to communicate from the given IP address to the SQL Database.

Error resolution: Enter the client's IP address as an IP rule. Do this by using the Firewall pane in the Azure portal.

Portal can create a virtual network rule

This section illustrates how you can use the [Azure portal](#) to create a *virtual network rule* in your Azure SQL Database. The rule tells your SQL Database to accept communication from a particular subnet that has been tagged as being a *Virtual Network service endpoint*.

NOTE

If you intend to add a service endpoint to the VNet firewall rules of your Azure SQL Database server, first ensure that service endpoints are turned On for the subnet.

If service endpoints are not turned on for the subnet, the portal asks you to enable them. Click the **Enable** button on the same blade on which you add the rule.

PowerShell alternative

A script can also create virtual network rules using PowerShell cmdlet **New-AzSqlServerVirtualNetworkRule** or **az network vnet create**. If interested, see [PowerShell to create a Virtual Network service endpoint and rule for Azure SQL Database](#).

REST API alternative

Internally, the PowerShell cmdlets for SQL VNet actions call REST APIs. You can call the REST APIs directly.

- [Virtual Network Rules: Operations](#)

Prerequisites

You must already have a subnet that is tagged with the particular Virtual Network service endpoint *type name* relevant to Azure SQL Database.

- The relevant endpoint type name is **Microsoft.Sql**.
- If your subnet might not be tagged with the type name, see [Verify your subnet is an endpoint](#).

Azure portal steps

1. Sign in to the [Azure portal](#).
2. Search for and select **SQL servers**, then select your server. Under **Security**, select **Firewalls and virtual networks**.
3. Set the **Allow access to Azure services** control to OFF.

IMPORTANT

If you leave the control set to ON, your Azure SQL Database server accepts communication from any subnet inside the Azure boundary i.e. originating from one of the IP addresses that is recognized as those within ranges defined for Azure data centers. Leaving the control set to ON might be excessive access from a security point of view. The Microsoft Azure Virtual Network service endpoint feature, in coordination with the virtual network rule feature of SQL Database, together can reduce your security surface area.

4. Click the **+ Add existing** control, in the **Virtual networks** section.



gm-sql-db-server-svr1 - Firewall / Virtual Networks

Save

Discard

Add client IP



Connections from the IPs specified below provides access to all the databases in gm-sql-db-server-svr1.

Allow access to Azure services

Client IP address 73.118.201.137

RULE NAME	START IP	END IP	...
			...
gm-ip-rule-ir1	172.27.26.0	172.27.26.255	...
gm-ip-rule-ir2	73.118.201.0	73.118.201.255	...



Connections from the VNET/Subnet specified below provides access to all databases in gm-sql-db-server-svr1.

Virtual networks

[+ Add existing](#)

[+ Create new](#)

RULE NAME	RESOURCE GROUP/VNET NAME	SUBNET

5. In the new **Create/Update** pane, fill in the controls with the names of your Azure resources.

TIP

You must include the correct **Address prefix** for your subnet. You can find the value in the portal. Navigate **All resources** > **All types** > **Virtual networks**. The filter displays your virtual networks. Click your virtual network, and then click **Subnets**. The **ADDRESS RANGE** column has the Address prefix you need.

@microsoft.co... MICROSOFT

Create/Update virtual network rule

* Name ⓘ gm-vnet-rule-vr42 ✓ provide vnet rule name

Subscription ⓘ no (8f) - ▾

* Virtual network ⓘ GM-VNet-20170907-a1

* Subnet name / Address prefix ⓘ subnet-a1-1 / 10.0.0.0/24

OK

The screenshot shows a Microsoft Azure portal dialog box titled "Create/Update virtual network rule". It contains several input fields: "Name" (gm-vnet-rule-vr42), "Subscription" (no (8f)), "Virtual network" (GM-VNet-20170907-a1), and "Subnet name / Address prefix" (subnet-a1-1 / 10.0.0.0/24). The "OK" button is visible at the bottom.

6. Click the **OK** button near the bottom of the pane.
7. See the resulting virtual network rule on the firewall pane.

Virtual networks	+ Add existing	+ Create new
RULE NAME	RESOURCE GROUP/VNET NAME	SUBNET
gm-vnet-rule-vr42	RG_GeneMi_22/GM-VNet-201709...	subnet-a... ...

NOTE

The following statuses or states apply to the rules:

- **Ready:** Indicates that the operation that you initiated has Succeeded.
- **Failed:** Indicates that the operation that you initiated has Failed.
- **Deleted:** Only applies to the Delete operation, and indicates that the rule has been deleted and no longer applies.
- **InProgress:** Indicates that the operation is in progress. The old rule applies while the operation is in this state.

Related articles

- [Azure virtual network service endpoints](#)
- [Azure SQL Database server-level and database-level firewall rules](#)

The virtual network rule feature for Azure SQL Database became available in late September 2017.

Next steps

- [Use PowerShell to create a virtual network service endpoint, and then a virtual network rule for Azure SQL Database.](#)
- [Virtual Network Rules: Operations with REST APIs](#)

PowerShell: Create a Virtual Service endpoint and VNet rule for SQL

11/22/2019 • 10 minutes to read • [Edit Online](#)

Virtual network rules are one firewall security feature that controls whether the database server for your single databases and elastic pool in Azure [SQL Database](#) or for your databases in [SQL Data Warehouse](#) accepts communications that are sent from particular subnets in virtual networks.

IMPORTANT

This article applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse. This article does *not* apply to a **managed instance** deployment in Azure SQL Database because it does not have a service endpoint associated with it.

This article provides and explains a PowerShell script that takes the following actions:

1. Creates a Microsoft Azure *Virtual Service endpoint* on your subnet.
2. Adds the endpoint to the firewall of your Azure SQL Database server, to create a *virtual network rule*.

Your motivations for creating a rule are explained in: [Virtual Service endpoints for Azure SQL Database](#).

TIP

If all you need is to assess or add the Virtual Service endpoint *type name* for SQL Database to your subnet, you can skip ahead to our more [direct PowerShell script](#).

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

Major cmdlets

This article emphasizes the **New-AzSqlServerVirtualNetworkRule** cmdlet that adds the subnet endpoint to the access control list (ACL) of your Azure SQL Database server, thereby creating a rule.

The following list shows the sequence of other *major* cmdlets that you must run to prepare for your call to **New-AzSqlServerVirtualNetworkRule**. In this article, these calls occur in [script 3 "Virtual network rule"](#):

1. [New-AzVirtualNetworkSubnetConfig](#): Creates a subnet object.
2. [New-AzVirtualNetwork](#): Creates your virtual network, giving it the subnet.
3. [Set-AzVirtualNetworkSubnetConfig](#): Assigns a Virtual Service endpoint to your subnet.
4. [Set-AzVirtualNetwork](#): Persists updates made to your virtual network.
5. [New-AzSqlServerVirtualNetworkRule](#): After your subnet is an endpoint, adds your subnet as a virtual network rule, into the ACL of your Azure SQL Database server.
 - This cmdlet Offers the parameter **-IgnoreMissingVNetServiceEndpoint**, starting in Azure RM PowerShell Module version 5.1.1.

Prerequisites for running PowerShell

- You can already log in to Azure, such as through the [Azure portal](#).
- You can already run PowerShell scripts.

NOTE

Please ensure that service endpoints are turned on for the VNet/Subnet that you want to add to your Server otherwise creation of the VNet Firewall Rule will fail.

One script divided into four chunks

Our demonstration PowerShell script is divided into a sequence of smaller scripts. The division eases learning and provides flexibility. The scripts must be run in their indicated sequence. If you do not have time now to run the scripts, our actual test output is displayed after script 4.

Script 1: Variables

This first PowerShell script assigns values to variables. The subsequent scripts depend on these variables.

IMPORTANT

Before you run this script, you can edit the values, if you like. For example, if you already have a resource group, you might want to edit your resource group name as the assigned value.

Your subscription name should be edited into the script.

PowerShell script 1 source code

```

#####
## Script 1 #####
## LOG into to your Azure account.      ##
## (Needed only one time per powershell.exe session.)  ##
#####

$yesno = Read-Host 'Do you need to log into Azure (only one time per powershell.exe session)? [yes/no]';  

if ('yes' -eq $yesno) { Connect-AzAccount; }

#####
## Assignments to variables used by the later scripts.  ##
#####

# You can edit these values, if necessary.  

$SubscriptionName = 'yourSubscriptionName';  

Select-AzSubscription -SubscriptionName $SubscriptionName;

$ResourceGroupName = 'RG-YourNameHere';  

$Region          = 'westcentralus';

$VNetName        = 'myVNet';  

$SubnetName       = 'mySubnet';  

$VNetAddressPrefix = '10.1.0.0/16';  

$SubnetAddressPrefix = '10.1.1.0/24';  

$VNetRuleName    = 'myFirstVNetRule-ForAcl';

$SqlDbServerName     = 'mysqlserver-forvnet';  

$SqlDbAdminLoginName = 'ServerAdmin';  

$SqlDbAdminLoginPassword = 'ChangeYourAdminPassword1';

$ServiceEndpointTypeName_SQLDB = 'Microsoft.Sql'; # Official type name.

Write-Host 'Completed script 1, the "Variables".';

```

Script 2: Prerequisites

This script prepares for the next script, where the endpoint action is. This script creates for you the following listed items, but only if they do not already exist. You can skip script 2 if you are sure these items already exist:

- Azure resource group
- Azure SQL Database server

PowerShell script 2 source code

```

#####
## Script 2 #####
## Ensure your Resource Group already exists.      ##
#####

Write-Host "Check whether your Resource Group already exists.";  
  

$gottenResourceGroup = $null;  
  

$gottenResourceGroup = Get-AzResourceGroup `  

    -Name      $ResourceGroupName `  

    -ErrorAction SilentlyContinue;  
  

if ($null -eq $gottenResourceGroup)  

{  

    Write-Host "Creating your missing Resource Group - $ResourceGroupName.";  
  

    $gottenResourceGroup = New-AzResourceGroup `  

        -Name $ResourceGroupName `  

        -Location $Region;  
  

    $gottenResourceGroup;  

}  

else { Write-Host "Good. your Resource Group already exists - $ResourceGroupName."; }

```

```

$gottenResourceGroup = $null;

#####
## Ensure your Azure SQL Database server already exists. ##
#####

Write-Host "Check whether your Azure SQL Database server already exists.';

$sqlDbServer = $null;

$sqlDbServer = Get-AzSqlServer `

-ResourceGroupName $ResourceGroupName `

-ServerName      $SqlDbServerName `

-ErrorAction     SilentlyContinue;

if ($null -eq $sqlDbServer) {
    Write-Host "Creating the missing Azure SQL Database server - $SqlDbServerName.";
    Write-Host "Gather the credentials necessary to next create an Azure SQL Database server.";

    $sqlAdministratorCredentials = New-Object `

-TypeName      System.Management.Automation.PSCredential `

-ArgumentList `

$SqlDbAdminLoginName, `

$(ConvertTo-SecureString `

-String        $SqlDbAdminLoginPassword `

-AsPlainText `

-Force `

);

if ($null -eq $sqlAdministratorCredentials) {
    Write-Host "ERROR, unable to create SQL administrator credentials. Now ending.";
    return;
}

Write-Host "Create your Azure SQL Database server.';

$sqlDbServer = New-AzSqlServer `

-ResourceGroupName $ResourceGroupName `

-ServerName      $SqlDbServerName `

-Location        $Region `

-SqlAdministratorCredentials $sqlAdministratorCredentials;

$sqlDbServer;

}
else {
    Write-Host "Good, your Azure SQL Database server already exists - $SqlDbServerName.";
}

$sqlAdministratorCredentials = $null;
$sqlDbServer               = $null;

Write-Host 'Completed script 2, the "Prerequisites".';

```

Script 3: Create an endpoint and a rule

This script creates a virtual network with a subnet. Then the script assigns the **Microsoft.Sql** endpoint type to your subnet. Finally the script adds your subnet to the access control list (ACL) of your SQL Database server, thereby creating a rule.

PowerShell script 3 source code

```

#####
## Script 3 #####
## Create your virtual network, and give it a subnet. ##
#####

```

```

Write-Host "Define a subnet '$SubnetName', to be given soon to a virtual network.";

$subnet = New-AzVirtualNetworkSubnetConfig ` 
    -Name          $SubnetName ` 
    -AddressPrefix $SubnetAddressPrefix ` 
    -ServiceEndpoint $ServiceEndpointTypeName_SQLDb;

Write-Host "Create a virtual network '$VNetName'." ` 
    " Give the subnet to the virtual network that we created.";

$vnet = New-AzVirtualNetwork ` 
    -Name          $VNetName ` 
    -AddressPrefix $VNetAddressPrefix ` 
    -Subnet        $subnet ` 
    -ResourceGroupName $ResourceGroupName ` 
    -Location      $Region;

#####
## Create a Virtual Service endpoint on the subnet. ## 
#####

Write-Host "Assign a Virtual Service endpoint 'Microsoft.Sql' to the subnet.";

$vnet = Set-AzVirtualNetworkSubnetConfig ` 
    -Name          $SubnetName ` 
    -AddressPrefix $SubnetAddressPrefix ` 
    -VirtualNetwork $vnet ` 
    -ServiceEndpoint $ServiceEndpointTypeName_SQLDb;

Write-Host "Persist the updates made to the virtual network > subnet.";

$vnet = Set-AzVirtualNetwork ` 
    -VirtualNetwork $vnet;

$vnet.Subnets[0].ServiceEndpoints; # Display the first endpoint.

#####
## Add the Virtual Service endpoint Id as a rule, ## 
## into SQL Database ACLs. ## 
#####

Write-Host "Get the subnet object.";

$vnet = Get-AzVirtualNetwork ` 
    -ResourceGroupName $ResourceGroupName ` 
    -Name          $VNetName;

$subnet = Get-AzVirtualNetworkSubnetConfig ` 
    -Name          $SubnetName ` 
    -VirtualNetwork $vnet;

Write-Host "Add the subnet .Id as a rule, into the ACLs for your Azure SQL Database server.";

$vnetRuleObject1 = New-AzSqlServerVirtualNetworkRule ` 
    -ResourceGroupName      $ResourceGroupName ` 
    -ServerName            $SqlDbServerName ` 
    -VirtualNetworkRuleName $VNetRuleName ` 
    -VirtualNetworkSubnetId $subnet.Id;

$vnetRuleObject1;

Write-Host "Verify that the rule is in the SQL DB ACL.";

$vnetRuleObject2 = Get-AzSqlServerVirtualNetworkRule ` 
    -ResourceGroupName      $ResourceGroupName ` 
    -ServerName            $SqlDbServerName ` 
    -VirtualNetworkRuleName $VNetRuleName;

```

```
$vnetRuleObject2;  
  
Write-Host 'Completed script 3, the "Virtual-Network-Rule".';
```

Script 4: Clean-up

This final script deletes the resources that the previous scripts created for the demonstration. However, the script asks for confirmation before it deletes the following:

- Azure SQL Database server
- Azure Resource Group

You can run script 4 any time after script 1 completes.

PowerShell script 4 source code

```

#####
## Script 4 #####
#### Clean-up phase A: Unconditional deletes. ##
## 1. The test rule is deleted from SQL DB ACL. ##
## 2. The test endpoint is deleted from the subnet. ##
## 3. The test virtual network is deleted. ##
#####

Write-Host "Delete the rule from the SQL DB ACL.";

Remove-AzSqlServerVirtualNetworkRule ` 
    -ResourceGroupName      $ResourceGroupName ` 
    -ServerName             $SqlDbServerName ` 
    -VirtualNetworkRuleName $VNetRuleName ` 
    -ErrorAction            SilentlyContinue;

Write-Host "Delete the endpoint from the subnet.";

$vnet = Get-AzVirtualNetwork ` 
    -ResourceGroupName $ResourceGroupName ` 
    -Name              $VNetName;

Remove-AzVirtualNetworkSubnetConfig ` 
    -Name $SubnetName ` 
    -VirtualNetwork $vnet;

Write-Host "Delete the virtual network (thus also deletes the subnet).";

Remove-AzVirtualNetwork ` 
    -Name          $VNetName ` 
    -ResourceGroupName $ResourceGroupName ` 
    -ErrorAction    SilentlyContinue;

#####
## Clean-up phase B: Conditional deletes. ##
## These might have already existed, so user might want to keep. ##
## 1. Azure SQL Database server ##
## 2. Azure resource group ##
#####

$yesno = Read-Host 'CAUTION !: Do you want to DELETE your Azure SQL Database server AND your Resource Group? [yes/no]';
if ('yes' -eq $yesno) {
    Write-Host "Remove the Azure SQL DB server.';

    Remove-AzSqlServer ` 
        -ServerName      $SqlDbServerName ` 
        -ResourceGroupName $ResourceGroupName ` 
        -ErrorAction     SilentlyContinue;

    Write-Host "Remove the Azure Resource Group.';

    Remove-AzResourceGroup ` 
        -Name          $ResourceGroupName ` 
        -ErrorAction   SilentlyContinue;
}

else {
    Write-Host "Skipped over the DELETE of SQL Database and resource group.";
}

Write-Host 'Completed script 4, the "Clean-Up".';

```

Verify your subnet is an endpoint

You might have a subnet that was already assigned the **Microsoft.Sql** type name, meaning it is already a Virtual Service endpoint. You could use the [Azure portal](#) to create a virtual network rule from the endpoint.

Or, you might be unsure whether your subnet has the **Microsoft.Sql** type name. You can run the following PowerShell script to take these actions:

1. Ascertain whether your subnet has the **Microsoft.Sql** type name.
2. Optionally, assign the type name if it is absent.
 - The script asks you to *confirm*, before it applies the absent type name.

Phases of the script

Here are the phases of the PowerShell script:

1. LOG into to your Azure account, needed only once per PS session. Assign variables.
2. Search for your virtual network, and then for your subnet.
3. Is your subnet tagged as **Microsoft.Sql** endpoint server type?
4. Add a Virtual Service endpoint of type name **Microsoft.Sql**, on your subnet.

IMPORTANT

Before you run this script, you must edit the values assigned to the \$-variables, near the top of the script.

Direct PowerShell source code

This PowerShell script does not update anything, unless you respond yes if it asks you for confirmation. The script can add the type name **Microsoft.Sql** to your subnet. But the script tries the add only if your subnet lacks the type name.

```
### 1. LOG into to your Azure account, needed only once per PS session. Assign variables.
$yesno = Read-Host 'Do you need to log into Azure (only one time per powershell.exe session)? [yes/no]';
if ('yes' -eq $yesno) { Connect-AzAccount; }

# Assignments to variables used by the later scripts.
# You can EDIT these values, if necessary.

$SubscriptionName = 'yourSubscriptionName';
Select-AzSubscription -SubscriptionName "$SubscriptionName";

$ResourceGroupName = 'yourRGName';
$VNetName          = 'yourVNetName';
$SubnetName         = 'yourSubnetName';
$SubnetAddressPrefix = 'Obtain this value from the Azure portal.'; # Looks roughly like: '10.0.0.0/24'

$ServiceEndpointTypeName_SQLDb = 'Microsoft.Sql'; # Do NOT edit. Is official value.

### 2. Search for your virtual network, and then for your subnet.
# Search for the virtual network.
$vnet = $null;
$vnet = Get-AzVirtualNetwork `

    -ResourceGroupName $ResourceGroupName `

    -Name              $VNetName;

if ($vnet -eq $null) {
    Write-Host "Caution: No virtual network found by the name '$VNetName'.";
    Return;
}

$subnet = $null;
for ($nn=0; $nn -lt $vnet.Subnets.Count; $nn++) {
```

```

$subnet = $vnet.Subnets[$nn];
if ($subnet.Name -eq $SubnetName)
{ break; }
$subnet = $null;
}

if ($subnet -eq $null) {
    Write-Host "Caution: No subnet found by the name '$SubnetName'";
    Return;
}

### 3. Is your subnet tagged as 'Microsoft.Sql' endpoint server type?
$endpointMsSql = $null;
for ($nn=0; $nn -lt $subnet.ServiceEndpoints.Count; $nn++) {
    $endpointMsSql = $subnet.ServiceEndpoints[$nn];
    if ($endpointMsSql.Service -eq $ServiceEndpointTypeName_SQLDb) {
        $endpointMsSql;
        break;
    }
    $endpointMsSql = $null;
}

if ($endpointMsSql -ne $null) {
    Write-Host "Good: Subnet found, and is already tagged as an endpoint of type
'$ServiceEndpointTypeName_SQLDb'.";
    Return;
}
else {
    Write-Host "Caution: Subnet found, but not yet tagged as an endpoint of type
'$ServiceEndpointTypeName_SQLDb'.;

    # Ask the user for confirmation.
    $yesno = Read-Host 'Do you want the PS script to apply the endpoint type name to your subnet? [yes/no]';
    if ('no' -eq $yesno) { Return; }
}

### 4. Add a Virtual Service endpoint of type name 'Microsoft.Sql', on your subnet.
$vnet = Set-AzVirtualNetworkSubnetConfig ` 
    -Name          $SubnetName ` 
    -AddressPrefix $SubnetAddressPrefix ` 
    -VirtualNetwork $vnet ` 
    -ServiceEndpoint $ServiceEndpointTypeName_SQLDb;

# Persist the subnet update.
$vnet = Set-AzVirtualNetwork ` 
    -VirtualNetwork $vnet;

for ($nn=0; $nn -lt $vnet.Subnets.Count; $nn++) {
    $vnet.Subnets[0].ServiceEndpoints; } # Display.

```

Private Link for Azure SQL Database and Data Warehouse (Preview)

1/16/2020 • 7 minutes to read • [Edit Online](#)

Private Link allows you to connect to various PaaS services in Azure via a **private endpoint**. For a list to PaaS services that support Private Link functionality, go to the [Private Link Documentation](#) page. A private endpoint is a private IP address within a specific [VNet](#) and Subnet.

IMPORTANT

This article applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse. This article does *not* apply to a **managed instance** deployment in Azure SQL Database.

Data exfiltration prevention

Data exfiltration in Azure SQL Database is when an authorized user, such as a database admin is able extract data from one system and move it another location or system outside the organization. For example, the user moves the data to a storage account owned by a third party.

Consider a scenario with a user running SQL Server Management Studio (SSMS) inside an Azure VM connecting to a SQL Database. This SQL Database is in the West US data center. The example below shows how to limit access with public endpoints on SQL Database using network access controls.

1. Disable all Azure service traffic to SQL Database via the public endpoint by setting Allow Azure Services to **OFF**. Ensure no IP addresses are allowed in the server and database level firewall rules. For more information, see [Azure SQL Database and Data Warehouse network access controls](#).
2. Only allow traffic to the SQL Database using the Private IP address of the VM. For more information, see the articles on [Service Endpoint](#) and [VNet firewall rules](#).
3. On the Azure VM, narrow down the scope of outgoing connection by using [Network Security Groups \(NSGs\)](#) and Service Tags as follows
 - Specify an NSG rule to allow traffic for Service Tag = SQL.WestUs - only allowing connection to SQL Database in West US
 - Specify an NSG rule (with a **higher priority**) to deny traffic for Service Tag = SQL - denying connections to SQL Database in all regions

At the end of this setup, the Azure VM can connect only to SQL Databases in the West US region. However, the connectivity isn't restricted to a single SQL Database. The VM can still connect to any SQL Databases in the West US region, including the databases that aren't part of the subscription. While we've reduced the scope of data exfiltration in the above scenario to a specific region, we haven't eliminated it altogether.

With Private Link, customers can now set up network access controls like NSGs to restrict access to the private endpoint. Individual Azure PaaS resources are then mapped to specific private endpoints. A malicious insider can only access the mapped PaaS resource (for example a SQL Database) and no other resource.

On-premises connectivity over private peering

When customers connect to the public endpoint from on-premises machines, their IP address needs to be added to the IP-based firewall using a [Server-level firewall rule](#). While this model works well for allowing access to

individual machines for dev or test workloads, it's difficult to manage in a production environment.

With Private Link, customers can enable cross-premises access to the private endpoint using ExpressRoute, private peering, or VPN tunneling. Customers can then disable all access via the public endpoint and not use the IP-based firewall to allow any IP addresses.

How to set up Private Link for Azure SQL Database

Creation Process

Private Endpoints can be created using the portal, PowerShell, or Azure CLI:

- [Portal](#)
- [PowerShell](#)
- [CLI](#)

Approval Process

Once the network admin creates the Private Endpoint (PE), the SQL admin can manage the Private Endpoint Connection (PEC) to SQL Database.

1. Navigate to the SQL server resource in the Azure portal as per steps shown in the screenshot below

- (1) Select the Private endpoint connections in the left pane
- (2) Shows a list of all Private Endpoint Connections (PECs)
- (3) Corresponding Private Endpoint (PE) created

CONNECTION NAME	STATE	PRIVATE ENDPOINT NAME	REQUEST/RESPONSE MESSAGE
myPEC1-ef9fd2df-df54-4dc4-9db2-978fe2d18a1b	Pending	myPEC1	
myPEC2-8950f0cd-cccd5-4a05-9661-53bcb4917028	Pending	myPEC2	

2. Select an individual PEC from the list by selecting it.

CONNECTION NAME	STATE	PRIVATE ENDPOINT NAME
myPEC1-ef9fd2df-df54-4dc4-9db2-978fe2d18a1b	Pending	myPEC1
myPEC2-8950f0cd-cccd5-4a05-9661-53bcb4917028	Pending	myPEC2

3. The SQL admin can choose to approve or reject a PEC and optionally add a short text response.

Approve
 Reject
 Remove

Approve connection

Do you want to approve the connection myPEC1-ef9fd2df-df54-4dc4-9db2-978fe2d18a1b?
 Request message: ""

Response message:

4. After approval or rejection, the list will reflect the appropriate state along with the response text.

Approve
 Reject
 Remove

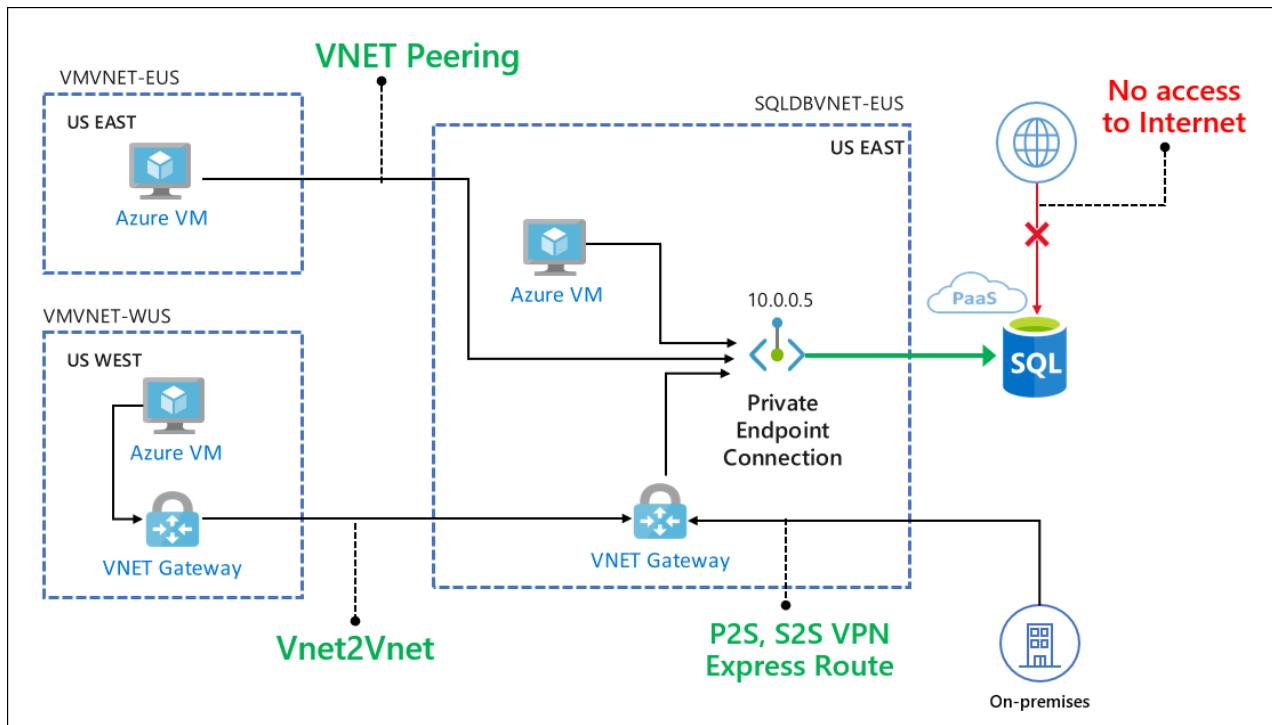
Private Endpoint Connection

Private endpoint connections allow connections from within a Virtual Network to a private IP using the private endpoint feature. Connections using these private endpoints specified below provide access to all databases in this

SEARCH...	3 selected	STATE	PRIVATE ENDPOINT NAME	REQUEST/RESPONSE MESSAGE
<input type="checkbox"/>	CONNECTION NAME	Approved	myPEC1	Approving myPEC1
<input type="checkbox"/>	myPEC1-ef9fd2df-df54-4dc4-9db2-978fe2d18a1b	Pending	myPEC2	

Use cases of Private Link for Azure SQL Database

Clients can connect to the Private endpoint from the same VNet, peered VNet in same region, or via VNet-to-VNet connection across regions. Additionally, clients can connect from on-premises using ExpressRoute, private peering, or VPN tunneling. Below is a simplified diagram showing the common use cases.



Test connectivity to SQL Database from an Azure VM in same Virtual Network (VNet)

For this scenario, assume you've created an Azure Virtual Machine (VM) running Windows Server 2016.

1. Start a Remote Desktop (RDP) session and connect to the virtual machine.
2. You can then do some basic connectivity checks to ensure that the VM is connecting to SQL Database via the

private endpoint using the following tools:

- a. Telnet
- b. Psping
- c. Nmap
- d. SQL Server Management Studio (SSMS)

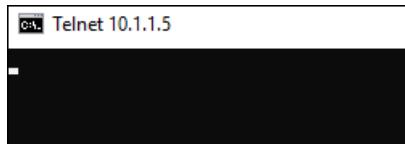
Check Connectivity using Telnet

[Telnet Client](#) is a Windows feature that can be used to test connectivity. Depending on the version of the Windows OS, you may need to enable this feature explicitly.

Open a Command Prompt window after you have installed Telnet. Run the Telnet command and specify the IP address and private endpoint of the SQL Database.

```
>telnet 10.1.1.5 1433
```

When Telnet connects successfully, you'll see a blank screen at the command window like the below image:



Check Connectivity using Psping

[Psping](#) can be used as follows to check that the Private endpoint connection(PEC) is listening for connections on port 1433.

Run psping as follows by providing the FQDN for your SQL Database server and port 1433:

```
>psping.exe mysqlsrvr.database.windows.net:1433

PsPing v2.10 - PsPing - ping, latency, bandwidth measurement utility
Copyright (C) 2012-2016 Mark Russinovich
Sysinternals - www.sysinternals.com

TCP connect to 10.6.1.4:1433:
5 iterations (warmup 1) ping test:
Connecting to 10.6.1.4:1433 (warmup): from 10.6.0.4:49953: 2.83ms
Connecting to 10.6.1.4:1433: from 10.6.0.4:49954: 1.26ms
Connecting to 10.6.1.4:1433: from 10.6.0.4:49955: 1.98ms
Connecting to 10.6.1.4:1433: from 10.6.0.4:49956: 1.43ms
Connecting to 10.6.1.4:1433: from 10.6.0.4:49958: 2.28ms
```

The output show that Psping could ping the private IP address associated with the PEC.

Check connectivity using Nmap

Nmap (Network Mapper) is a free and open-source tool used for network discovery and security auditing. For more information and the download link, visit <https://nmap.org>. You can use this tool to ensure that the private endpoint is listening for connections on port 1433.

Run Nmap as follows by providing the address range of the subnet that hosts the private endpoint.

```
>nmap -n -sP 10.1.1.0/24
...
...
Nmap scan report for 10.1.1.5
Host is up (0.00s latency).
Nmap done: 256 IP addresses (1 host up) scanned in 207.00 seconds
```

The result shows that one IP address is up; which corresponds to the IP address for the private endpoint.

Check Connectivity using SQL Server Management Studio (SSMS)

NOTE

Use the **Fully Qualified Domain Name(FQDN)** of the server in connection strings for your clients. Any login attempts made directly to the IP address shall fail by design.

Follow the steps here to use [SSMS to connect to the SQL Database](#). After you connect to the SQL Database using SSMS, verify that you're connecting from the private IP address of the Azure VM by running the following query:

```
select client_net_address from sys.dm_exec_connections  
where session_id=@@SPID
```

NOTE

In preview, connections to private endpoint only support **Proxy** as the [connection policy](#)

Connecting from an Azure VM in Peered Virtual Network (VNet)

Configure [VNet peering](#) to establish connectivity to the SQL Database from an Azure VM in a peered VNet.

Connecting from an Azure VM in VNet-to-VNet environment

Configure [VNet-to-VNet VPN gateway connection](#) to establish connectivity to a SQL Database from an Azure VM in a different region or subscription.

Connecting from an on-premises environment over VPN

To establish connectivity from an on-premises environment to the SQL Database, choose and implement one of the options:

- [Point-to-Site connection](#)
- [Site-to-Site VPN connection](#)
- [ExpressRoute circuit](#)

Connecting from an Azure SQL Data Warehouse to Azure Storage using Polybase

PolyBase is commonly used to load data into Azure SQL Data Warehouse from Azure Storage accounts. If the Azure Storage account that you are loading data from limits access only to a set of VNet-subnets via Private Endpoints, Service Endpoints, or IP-based firewalls, the connectivity from PolyBase to the account will break. For enabling both PolyBase import and export scenarios with Azure SQL Data Warehouse connecting to Azure Storage that's secured to a VNet, follow the steps provided [here](#).

Next steps

- For an overview of Azure SQL Database security, see [Securing your database](#)
- For an overview of Azure SQL Database connectivity, see [Azure SQL Connectivity Architecture](#)

SQL Database Audit Log Format

1/23/2020 • 4 minutes to read • [Edit Online](#)

Azure SQL Database auditing tracks database events and writes them to an audit log in your Azure storage account, or sends them to Event Hub or Log Analytics for downstream processing and analysis.

Naming Conventions

Blob Audit

Audit logs stored in Blob storage are stored in a container named `sqlbauditlogs` in the Azure Storage account. The directory hierarchy within the container is of the form `<ServerName>/<DatabaseName>/<AuditName>/<Date>/`. The Blob filename format is `<CreationTime>_<FileNumberInSession>.xel`, where `CreationTime` is in UTC `hh_mm_ss_ms` format, and `FileNumberInSession` is a running index in case session logs spans across multiple Blob files.

For example, for database `Database1` on `Server1` the following is a possible valid path:

```
Server1/Database1/SqlDbAuditing_ServerAudit_NoRetention/2019-02-03/12_23_30_794_0.xel
```

Read-only Replicas Audit logs are stored in the same container. The directory hierarchy within the container is of the form `<ServerName>/<DatabaseName>/<AuditName>/<Date>/RO/`. The Blob filename shares the same format. The Audit Logs of Read-only Replicas are stored in the same container.

Event Hub

Audit events are written to the namespace and event hub that was defined during auditing configuration, and are captured in the body of [Apache Avro](#) events and stored using JSON formatting with UTF-8 encoding. To read the audit logs, you can use [Avro Tools](#) or similar tools that process this format.

Log Analytics

Audit events are written to Log Analytics workspace defined during auditing configuration, to the `AzureDiagnostics` table with the category `SQLSecurityAuditEvents`. For additional useful information about Log Analytics search language and commands, see [Log Analytics search reference](#).

Audit Log Fields

NAME (BLOB)	NAME (EVENT HUBS/LOG ANALYTICS)	DESCRIPTION	BLOB TYPE	EVENT HUBS/LOG ANALYTICS TYPE
action_id	action_id_s	ID of the action	varchar(4)	string
action_name	action_name_s	Name of the action	N/A	string
additional_information	additional_information_s	Any additional information about the event, stored as XML	nvarchar(4000)	string
affected_rows	affected_rows_d	Number of rows affected by the query	bigint	int

NAME (BLOB)	NAME (EVENT HUBS/LOG ANALYTICS)	DESCRIPTION	BLOB TYPE	EVENT HUBS/LOG ANALYTICS TYPE
application_name	application_name_s	Name of client application	nvarchar(128)	string
audit_schema_version	audit_schema_version_d	Always 1	int	int
class_type	class_type_s	Type of auditable entity that the audit occurs on	varchar(2)	string
class_type_desc	class_type_description_s	Description of auditable entity that the audit occurs on	N/A	string
client_ip	client_ip_s	Source IP of the client application	nvarchar(128)	string
connection_id	N/A	ID of the connection in the server	GUID	N/A
data_sensitivity_information	data_sensitivity_information_s	Information types and sensitivity labels returned by the audited query, based on the classified columns in the database. Learn more about Azure SQL Database data discover and classification	nvarchar(4000)	string
database_name	database_name_s	The database context in which the action occurred	sysname	string
database_principal_id	database_principal_id_d	ID of the database user context that the action is performed in	int	int
database_principal_name	database_principal_name_s	Name of the database user context in which the action is performed	sysname	string
duration_milliseconds	durationMilliseconds_d	Query execution duration in milliseconds	bigint	int
event_time	event_time_t	Date and time when the auditable action is fired	datetime2	datetime
host_name	N/A	Client host name	string	N/A

NAME (BLOB)	NAME (EVENT HUBS/LOG ANALYTICS)	DESCRIPTION	BLOB TYPE	EVENT HUBS/LOG ANALYTICS TYPE
is_column_permission	is_column_permission_s	Flag indicating if this is a column level permission. 1 = true, 0 = false	bit	string
N/A	is_server_level_audit_s	Flag indicating if this audit is at the server level	N/A	string
object_id	object_id_d	The ID of the entity on which the audit occurred. This includes the : server objects, databases, database objects, and schema objects. 0 if the entity is the server itself or if the audit is not performed at an object level	int	int
object_name	object_name_s	The name of the entity on which the audit occurred. This includes the : server objects, databases, database objects, and schema objects. 0 if the entity is the server itself or if the audit is not performed at an object level	sysname	string
permission_bitmask	permission_bitmask_s	When applicable, shows the permissions that were granted, denied, or revoked	varbinary(16)	string
response_rows	response_rows_d	Number of rows returned in the result set	bigint	int
schema_name	schema_name_s	The schema context in which the action occurred. NULL for audits occurring outside a schema	sysname	string
N/A	securable_class_type_s	Securable object that maps to the class_type being audited	N/A	string
sequence_group_id	sequence_group_id_g	Unique identifier	varbinary	GUID

NAME (BLOB)	NAME (EVENT HUBS/LOG ANALYTICS)	DESCRIPTION	BLOB TYPE	EVENT HUBS/LOG ANALYTICS TYPE
sequence_number	sequence_number_d	Tracks the sequence of records within a single audit record that was too large to fit in the write buffer for audits	int	int
server_instance_name	server_instance_name_s	Name of the server instance where the audit occurred	sysname	string
server_principal_id	server_principal_id_d	ID of the login context in which the action is performed	int	int
server_principal_name	server_principal_name_s	Current login	sysname	string
server_principal_sid	server_principal_sid_s	Current login SID	varbinary	string
session_id	session_id_d	ID of the session on which the event occurred	smallint	int
session_server_principal_name	session_server_principal_name_s	Server principal for session	sysname	string
statement	statement_s	T-SQL statement that was executed (if any)	nvarchar(4000)	string
succeeded	succeeded_s	Indicates whether the action that triggered the event succeeded. For events other than login and batch, this only reports whether the permission check succeeded or failed, not the operation. 1 = success, 0 = fail	bit	string
target_database_principal_id	target_database_principal_id_d	The database principal the GRANT/DENY/REVOKE operation is performed on. 0 if not applicable	int	int
target_database_principal_name	target_database_principal_name_s	Target user of action. NULL if not applicable	string	string

NAME (BLOB)	NAME (EVENT HUBS/LOG ANALYTICS)	DESCRIPTION	BLOB TYPE	EVENT HUBS/LOG ANALYTICS TYPE
target_server_principal_id	target_server_principal_id_d	Server principal that the GRANT/DENY/REVOKE operation is performed on. Returns 0 if not applicable	int	int
target_server_principal_name	target_server_principal_name_s	Target login of action. NULL if not applicable	sysname	string
target_server_principal_sid	target_server_principal_sid_s	SID of target login. NULL if not applicable	varbinary	string
transaction_id	transaction_id_d	SQL Server only (starting with 2016) - 0 for Azure SQL DB	bigint	int
user_defined_event_id	user_defined_event_id_d	User defined event id passed as an argument to sp_audit_write. NULL for system events (default) and non-zero for user-defined event. For more information, see sp_audit_write (Transact-SQL)	smallint	int
user_defined_information	user_defined_information_s	User defined information passed as an argument to sp_audit_write. NULL for system events (default) and non-zero for user-defined event. For more information, see sp_audit_write (Transact-SQL)	nvarchar(4000)	string

Next Steps

Learn more about [Azure SQL Database auditing](#).

Conditional Access (MFA) with Azure SQL Database and Data Warehouse

11/7/2019 • 2 minutes to read • [Edit Online](#)

Azure [SQL Database](#), [Managed Instance](#), and [SQL Data Warehouse](#) support Microsoft Conditional Access.

NOTE

This topic applies to Azure SQL server, and to both SQL Database and SQL Data Warehouse databases that are created on the Azure SQL server. For simplicity, SQL Database is used when referring to both SQL Database and SQL Data Warehouse.

The following steps show how to configure SQL Database to enforce a Conditional Access policy.

Prerequisites

- You must configure your SQL Database or SQL Data Warehouse to support Azure Active Directory authentication. For specific steps, see [Configure and manage Azure Active Directory authentication with SQL Database or SQL Data Warehouse](#).
- When multi-factor authentication is enabled, you must connect with at supported tool, such as the latest SSMS. For more information, see [Configure Azure SQL Database multi-factor authentication for SQL Server Management Studio](#).

Configure CA for Azure SQL DB/DW

1. Sign in to the Portal, select **Azure Active Directory**, and then select **Conditional Access**. For more information, see [Azure Active Directory Conditional Access technical reference](#).

Microsoft Azure azcatest

azcatest
Azure Active Directory

Classic portal Switch directory Delete directory

Users and groups

Enterprise applications

Enterprise applications

Sync enabled

Recommended

Sync with Windows Server AD

Sync users and groups from your on-premises directory to your Azure AD

App registrations

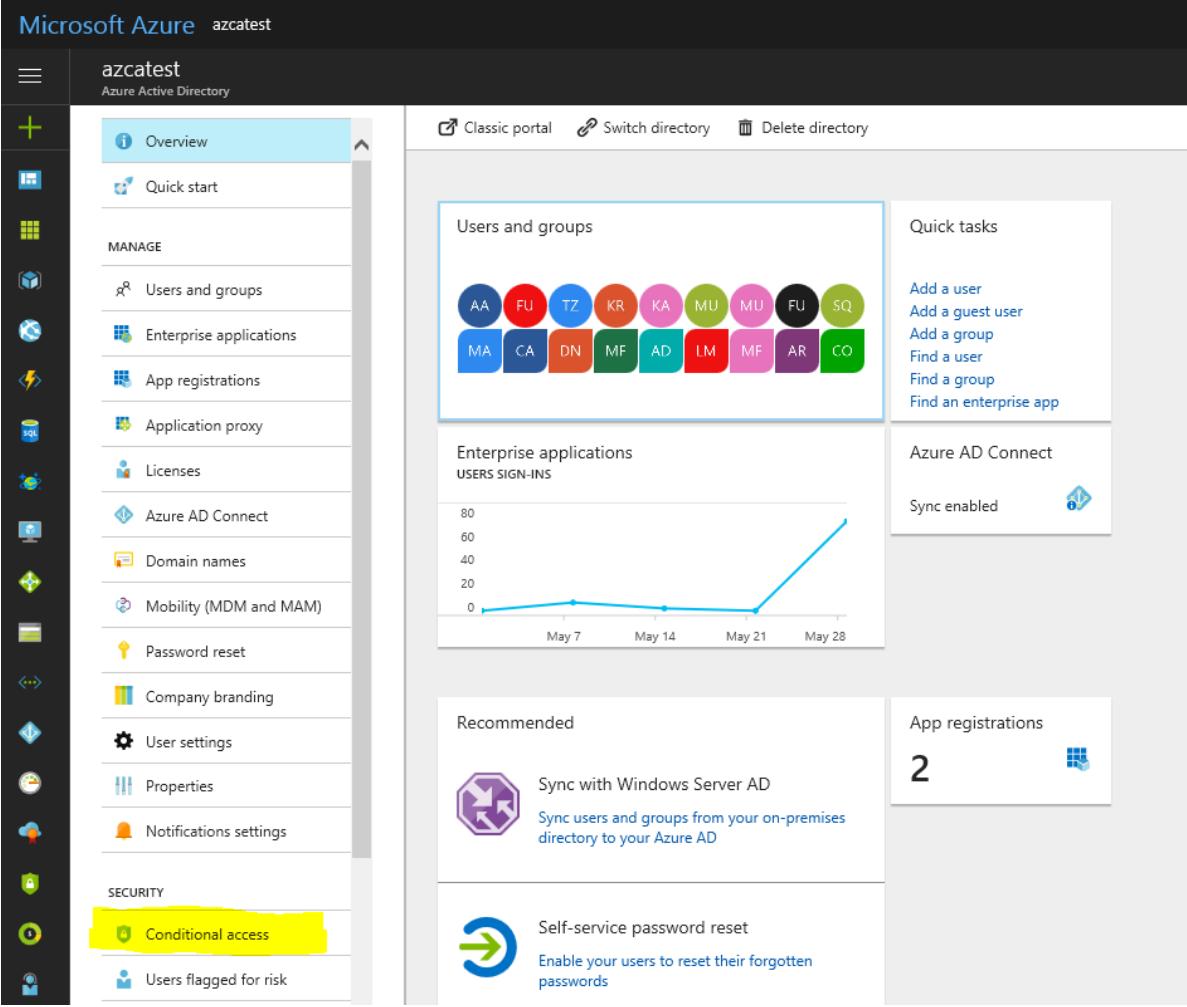
2

Self-service password reset

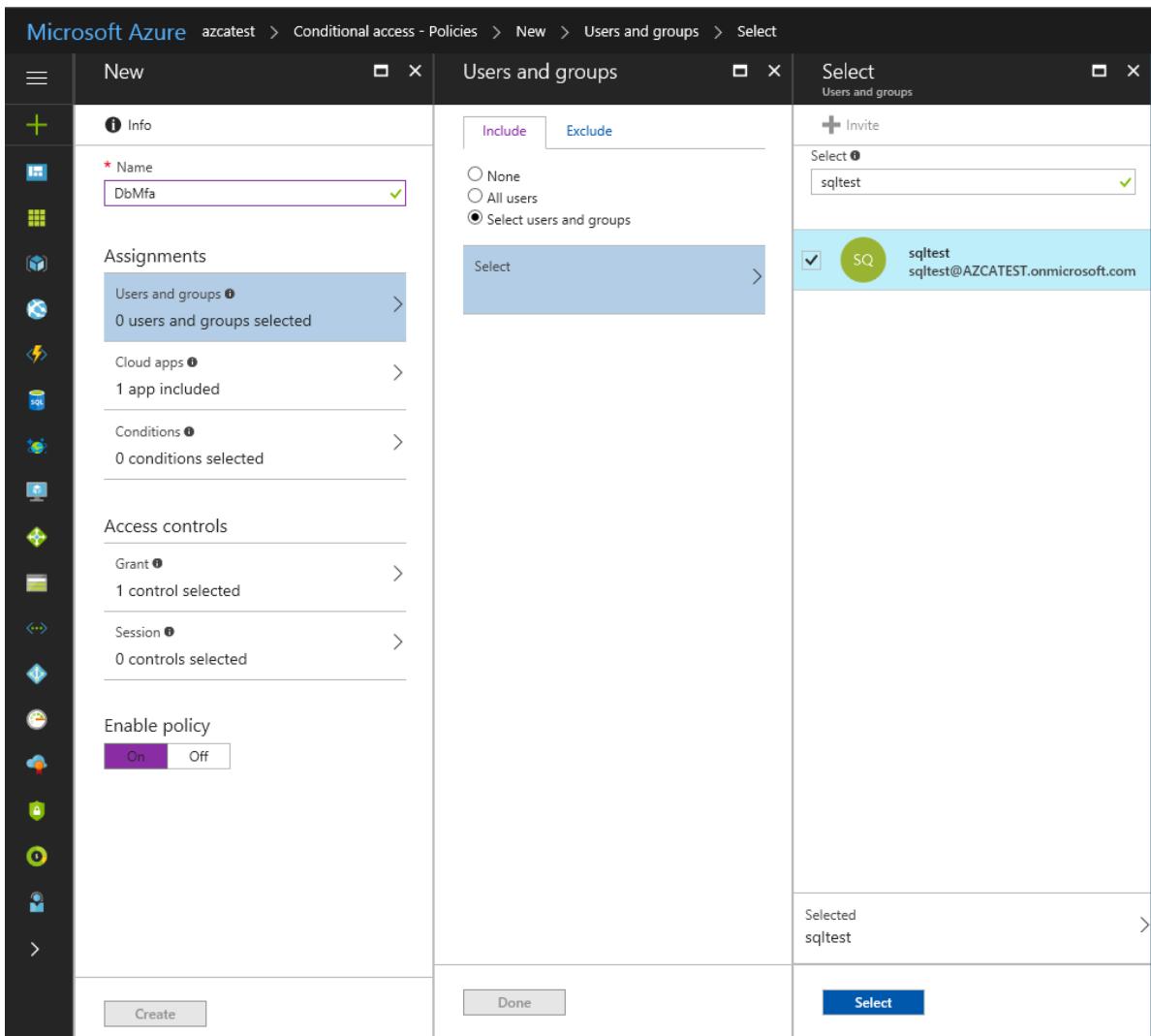
Enable your users to reset their forgotten passwords

Conditional access

Users flagged for risk



2. In the **Conditional Access-Policies** blade, click **New policy**, provide a name, and then click **Configure rules**.
3. Under **Assignments**, select **Users and groups**, check **Select users and groups**, and then select the user or group for Conditional Access. Click **Select**, and then click **Done** to accept your selection.



4. Select **Cloud apps**, click **Select apps**. You see all apps available for Conditional Access. Select **Azure SQL Database**, at the bottom click **Select**, and then click **Done**.

The screenshot shows the Microsoft Azure Conditional access - Policies - New - Cloud apps interface. On the left, there's a sidebar with various icons. The main area has tabs for 'Info', 'Assignments', 'Access controls', and 'Enable policy'. Under 'Assignments', 'Cloud apps' is selected. Under 'Access controls', 'Grant' is selected. Under 'Enable policy', 'On' is chosen. At the bottom are 'Create' and 'Done' buttons.

If you can't find **Azure SQL Database** listed in the following third screenshot, complete the following steps:

- Sign in to your Azure SQL DB/DW instance using SSMS with an AAD admin account.
 - Execute `CREATE USER [user@yourtenant.com] FROM EXTERNAL PROVIDER`.
 - Sign in to AAD and verify that Azure SQL Database and Data Warehouse are listed in the applications in your AAD.
5. Select **Access controls**, select **Grant**, and then check the policy you want to apply. For this example, we select **Require multi-factor authentication**.

Microsoft Azure azcatest > Conditional access - Policies > New > Grant

The screenshot shows the Microsoft Azure Conditional Access Policy creation interface. On the left, a sidebar lists various policy types with icons. The 'Grant' control is selected, indicated by a blue background. The main area is divided into two tabs: 'New' (left) and 'Grant' (right). The 'Grant' tab is active, showing the configuration for the selected control.

Info

* Name: DbMfa

Assignments

- Users and groups: 1 user included
- Cloud apps: 1 app included
- Conditions: 0 conditions selected

Access controls

- Grant: 1 control selected
- Session: 0 controls selected

Enable policy

On

Grant

Select the controls to be enforced.

Block access
 Grant access

Require multi-factor authentication

Require device to be marked as compliant

Require domain joined device

For multiple controls

Require all the selected controls
 Require one of the selected controls (preview)

Create Select

Summary

The selected application (Azure SQL Database) allowing to connect to Azure SQL DB/DW using Azure AD Premium, now enforces the selected Conditional Access policy, **Required multi-factor authentication**.
For questions about Azure SQL Database and Data Warehouse regarding multi-factor authentication, contact MFAforSQLDB@microsoft.com.

Next steps

For a tutorial, see [Secure your Azure SQL Database](#).

Use an Azure SQL Database managed instance securely with public endpoints

11/7/2019 • 2 minutes to read • [Edit Online](#)

Azure SQL Database managed instances can provide user connectivity over [public endpoints](#). This article explains how to make this configuration more secure.

Scenarios

A SQL Database managed instance provides a private endpoint to allow connectivity from inside its virtual network. The default option is to provide maximum isolation. However, there are scenarios where you need to provide a public endpoint connection:

- The managed instance must integrate with multi-tenant-only platform-as-a-service (PaaS) offerings.
- You need higher throughput of data exchange than is possible when you're using a VPN.
- Company policies prohibit PaaS inside corporate networks.

Deploy a managed instance for public endpoint access

Although not mandatory, the common deployment model for a managed instance with public endpoint access is to create the instance in a dedicated isolated virtual network. In this configuration, the virtual network is used only for virtual cluster isolation. It doesn't matter if the managed instance's IP address space overlaps with a corporate network's IP address space.

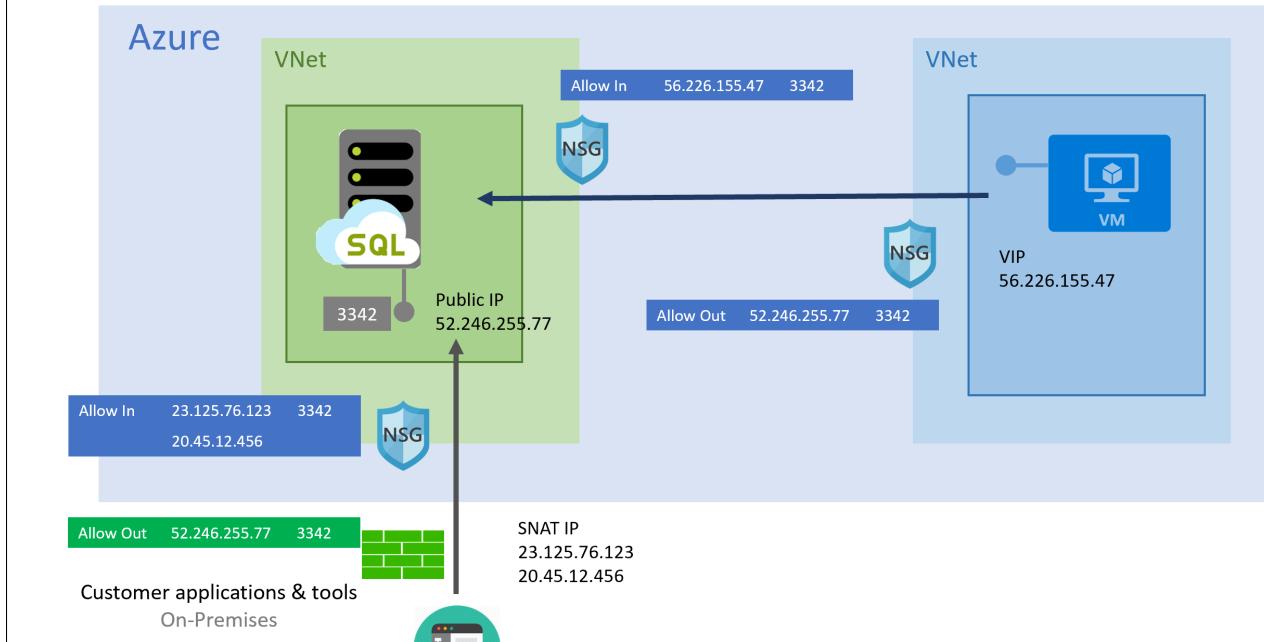
Secure data in motion

Managed instance data traffic is always encrypted if the client driver supports encryption. Data sent between the managed instance and other Azure virtual machines or Azure services never leaves Azure's backbone. If there's a connection between the managed instance and an on-premises network, we recommend you use Azure ExpressRoute with Microsoft peering. ExpressRoute helps you avoid moving data over the public internet. For managed instance private connectivity, only private peering can be used.

Lock down inbound and outbound connectivity

The following diagram shows the recommended security configurations:

Public Endpoint - Secure Access



A managed instance has a [dedicated public endpoint address](#). In the client-side outbound firewall and in the network security group rules, set this public endpoint IP address to limit outbound connectivity.

To ensure traffic to the managed instance is coming from trusted sources, we recommend connecting from sources with well-known IP addresses. Use a network security group to limit access to the managed instance public endpoint on port 3342.

When clients need to initiate a connection from an on-premises network, make sure the originating address is translated to a well-known set of IP addresses. If you can't do so (for example, a mobile workforce being a typical scenario), we recommend you use [point-to-site VPN connections and a private endpoint](#).

If connections are started from Azure, we recommend that traffic come from a well-known assigned [virtual IP address](#) (for example, a virtual machine). To make managing virtual IP (VIP) addresses easier, you might want to use [public IP address prefixes](#).

Next steps

- Learn how to configure public endpoint for manage instances: [Configure public endpoint](#)

Get started with Azure SQL Database managed instance auditing

1/23/2020 • 7 minutes to read • [Edit Online](#)

Managed instance auditing tracks database events and writes them to an audit log in your Azure storage account. Auditing also:

- Helps you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations.
- Enables and facilitates adherence to compliance standards, although it doesn't guarantee compliance. For more information about Azure programs that support standards compliance, see the [Azure Trust Center](#) where you can find the most current list of SQL Database compliance certifications.

Set up auditing for your server to Azure storage

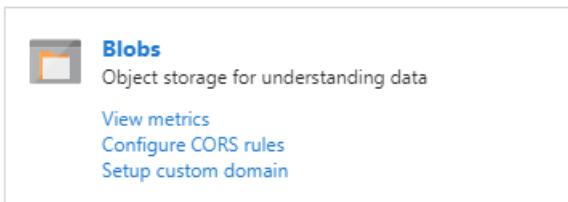
The following section describes the configuration of auditing on your managed instance.

1. Go to the [Azure portal](#).
2. Create an Azure Storage **container** where audit logs are stored.
 - a. Navigate to the Azure Storage where you would like to store your audit logs.

IMPORTANT

Use a storage account in the same region as the managed instance to avoid cross-region reads/writes.

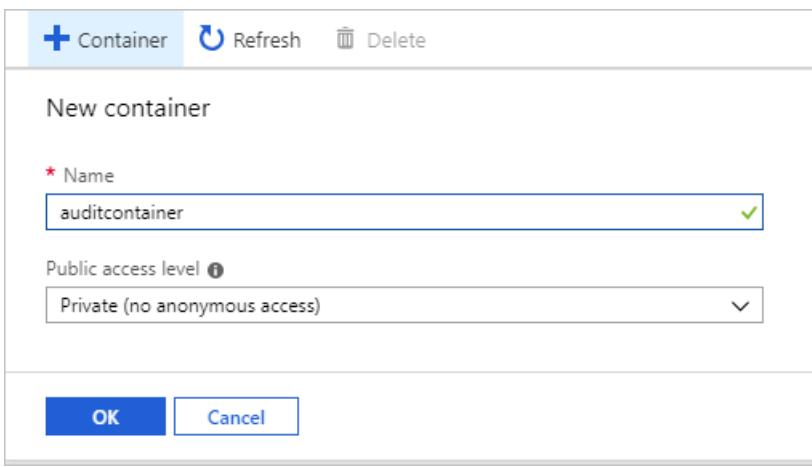
- a. In the storage account, go to **Overview** and click **Blobs**.



- c. In the top menu, click **+ Container** to create a new container.

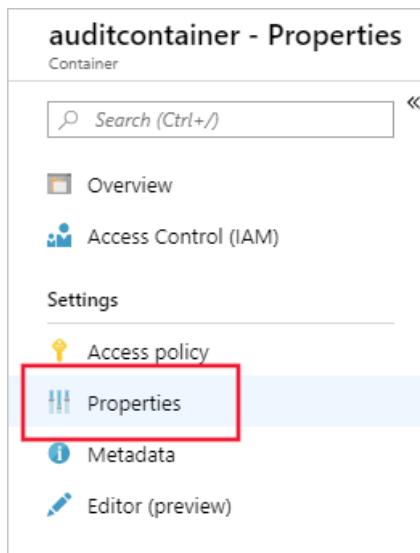


- d. Provide a container **Name**, set Public access level to **Private**, and then click **OK**.



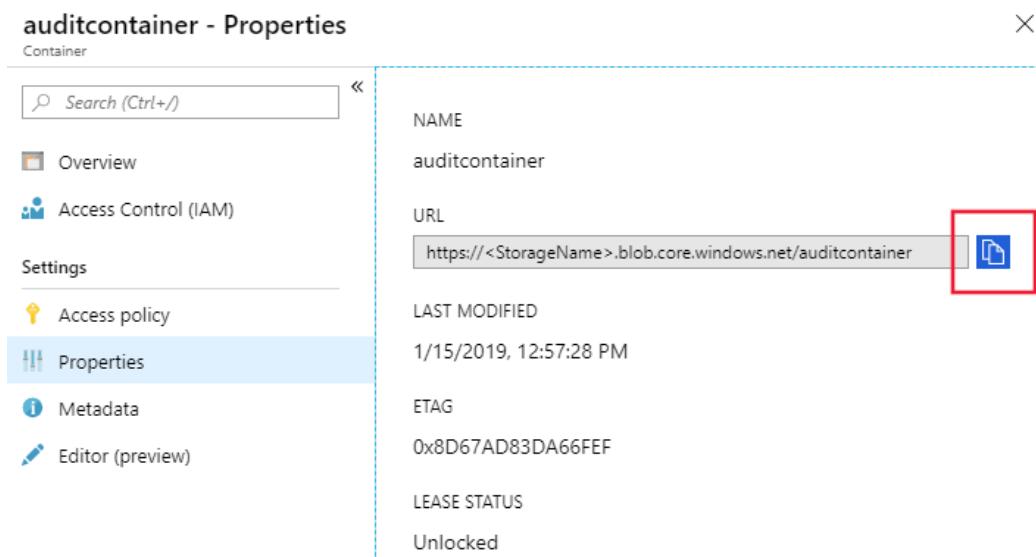
3. After creating the container for the Audit logs there are two ways to configure it as the target for the audit logs: [using T-SQL](#) or [using the SQL Server Management Studio \(SSMS\) UI](#):

- Configure blob storage for audit logs using T-SQL:
 - In the containers list, click the newly created container and then click **Container properties**.



- Copy the container URL by clicking the copy icon and save the URL (for example, in Notepad) for future use. The container URL format should be

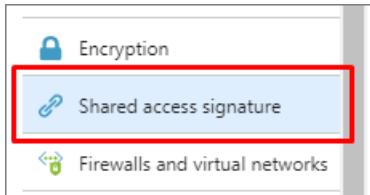
```
https://<StorageName>.blob.core.windows.net/<ContainerName>
```



- Generate an Azure Storage **SAS Token** to grant managed instance auditing access rights to

the storage account:

- o Navigate to the Azure Storage account where you created the container in the previous step.
- o Click on **Shared access signature** in the Storage Settings menu.



- o Configure the SAS as follows:
 - o **Allowed services:** Blob
 - o **Start date:** to avoid time zone-related issues, it is recommended to use yesterday's date
 - o **End date:** choose the date on which this SAS Token expires

NOTE

Renew the token upon expiry to avoid audit failures.

- o Click **Generate SAS**.

The dialog box contains the following configuration options:

- Allowed services:** Blob, File, Queue, Table
- Allowed resource types:** Service, Container, Object
- Allowed permissions:** Read, Write, Delete, List, Add, Create, Update, Process
- Start and expiry date/time:**
 - Start:** 2018-03-18 12:03:39 PM
 - End:** 2018-03-19 8:03:39 PM
(UTC+02:00) --- Current Timezone ---
- Allowed IP addresses:** for example, 168.1.5.65 or 168.1.5.65-168.1.5.70
- Allowed protocols:** HTTPS only, HTTPS and HTTP
- Signing key:** key1
- Generate SAS** button

- o After clicking on Generate SAS, the SAS Token appears at the bottom. Copy the token by clicking on the copy icon, and save it (for example, in Notepad) for future use.



IMPORTANT

Remove the question mark ("?") character from the beginning of the token.

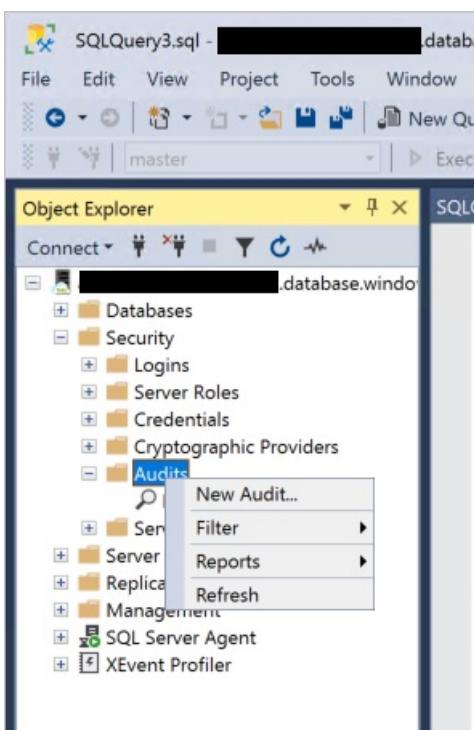
- d. Connect to your managed instance via SQL Server Management Studio (SSMS) or any other supported tool.
- e. Execute the following T-SQL statement to **create a new Credential** using the Container URL and SAS Token that you created in the previous steps:

```
CREATE CREDENTIAL [<container_url>]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = '<SAS KEY>'
GO
```

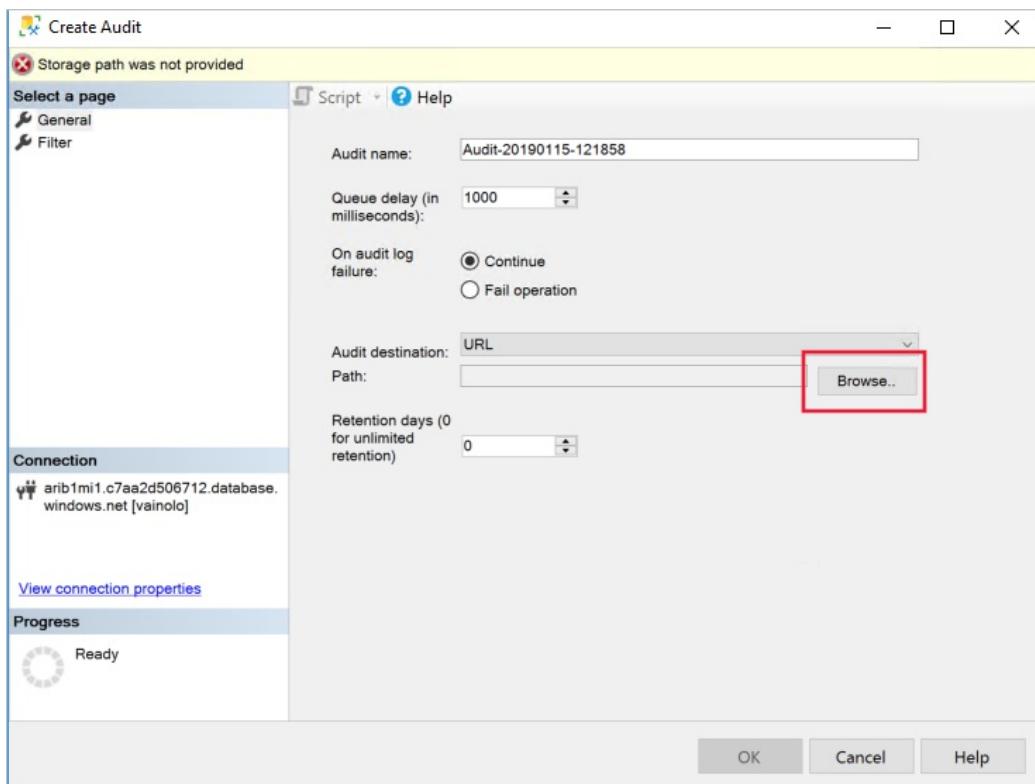
- f. Execute the following T-SQL statement to create a new Server Audit (choose your own audit name, use the Container URL that you created in the previous steps). If not specified, **RETENTION_DAYS** default is 0 (unlimited retention):

```
CREATE SERVER AUDIT [<your_audit_name>]
TO URL ( PATH = '<container_url>' [, RETENTION_DAYS = integer ])
GO
```

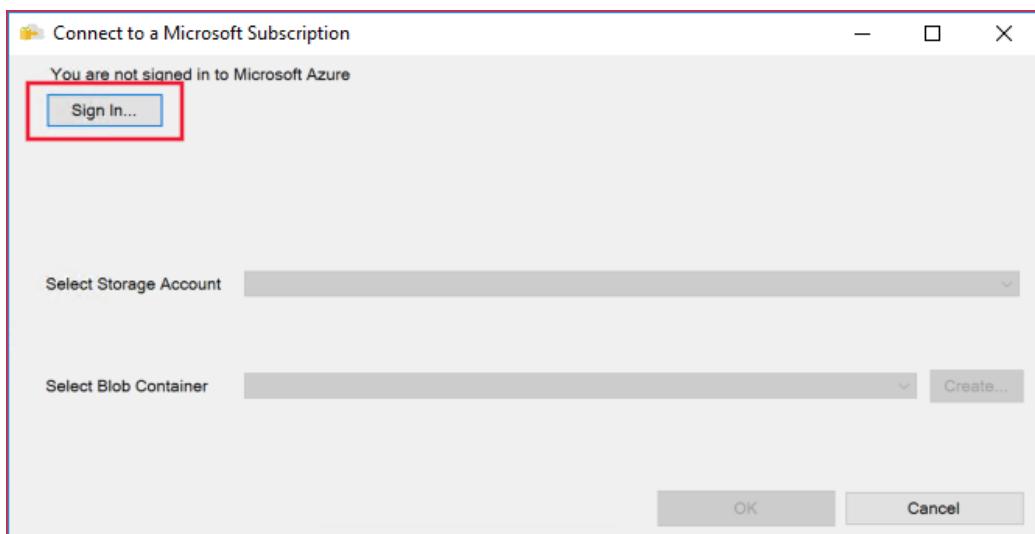
- a. Continue by [creating a Server Audit Specification or Database Audit Specification](#)
- Configure blob storage for audit logs using the SQL Server Management Studio (SSMS) 18 (Preview):
 - a. Connect to the managed instance using SQL Server Management Studio (SSMS) UI.
 - b. Expand the root note of the Object Explorer.
 - c. Expand the **Security** node, right-click on the **Audits** node, and click on "New Audit":



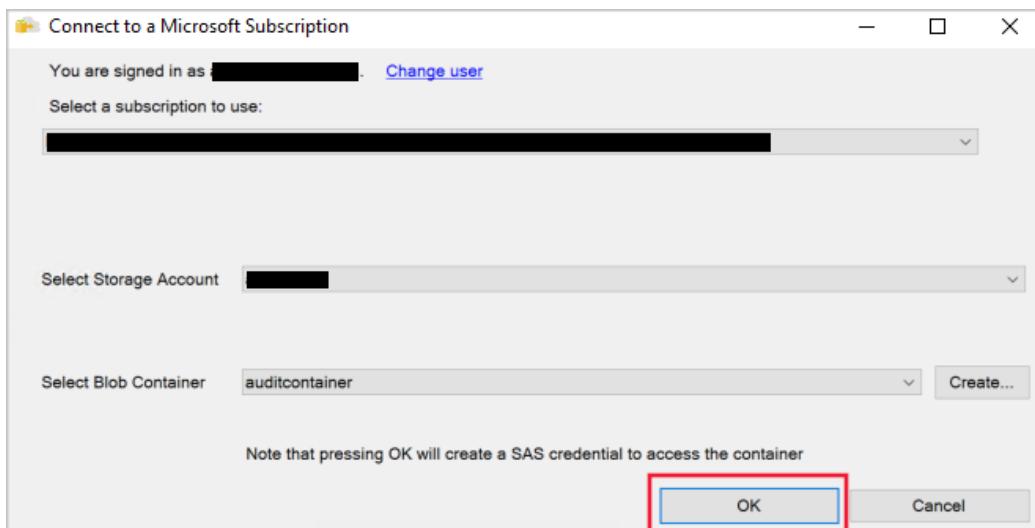
- d. Make sure "URL" is selected in **Audit destination** and click on **Browse**:



e. (Optional) Sign in to your Azure account:



f. Select a subscription, storage account, and Blob container from the dropdowns, or create your own container by clicking on **Create**. Once you have finished click **OK**:



- g. Click **OK** in the "Create Audit" dialog.
4. After configuring the Blob container as target for the audit logs, create a Server Audit Specification or Database Audit Specification as you would for SQL Server:
- [Create Server audit specification T-SQL guide](#)
 - [Create Database audit specification T-SQL guide](#)
5. Enable the server audit that you created in step 6:

```
ALTER SERVER AUDIT [<your_audit_name>]
WITH (STATE=ON);
GO
```

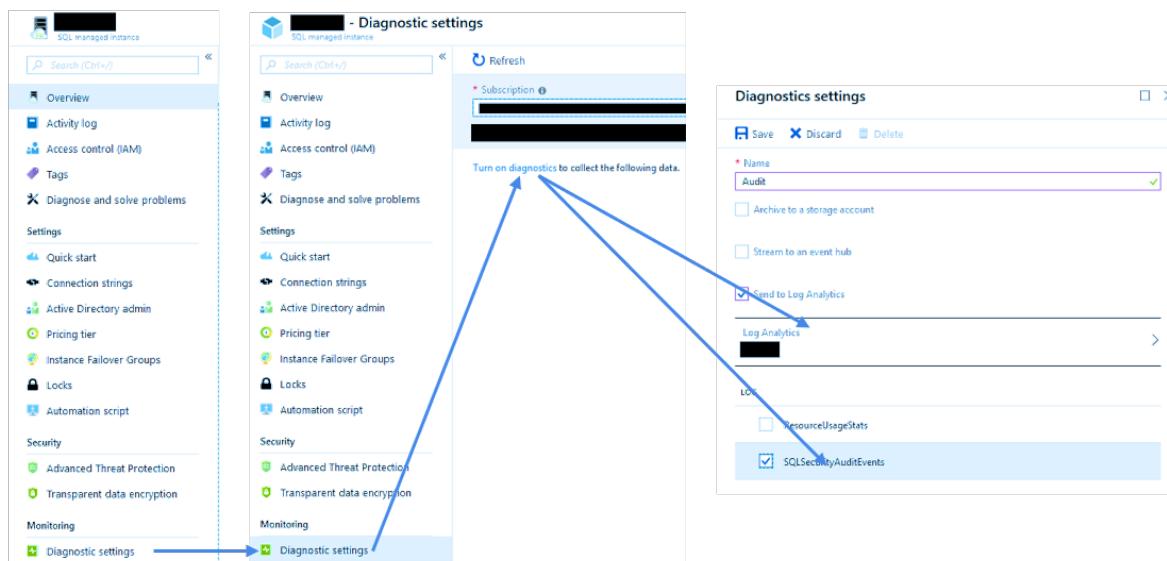
For additional information:

- [Auditing differences between single databases, elastic pools, and managed instances in Azure SQL Database and databases in SQL Server](#)
- [CREATE SERVER AUDIT](#)
- [ALTER SERVER AUDIT](#)

Set up auditing for your server to Event Hub or Azure Monitor logs

Audit logs from a managed instance can be sent to Event Hubs or Azure Monitor logs. This section describes how to configure this:

1. Navigate in the [Azure Portal](#) to the managed instance.
2. Click on **Diagnostic settings**.
3. Click on **Turn on diagnostics**. If diagnostics is already enabled the *+Add diagnostic setting* will show instead.
4. Select **SQLSecurityAuditEvents** in the list of logs.
5. Select a destination for the audit events - Event Hub, Azure Monitor logs, or both. Configure for each target the required parameters (e.g. Log Analytics workspace).
6. Click **Save**.



7. Connect to the managed instance using **SQL Server Management Studio (SSMS)** or any other supported client.

8. Execute the following T-SQL statement to create a server audit:

```
CREATE SERVER AUDIT [<your_audit_name>] TO EXTERNAL_MONITOR;
GO
```

9. Create a server audit specification or database audit specification as you would for SQL Server:

- [Create Server audit specification T-SQL guide](#)
- [Create Database audit specification T-SQL guide](#)

10. Enable the server audit created in step 8:

```
ALTER SERVER AUDIT [<your_audit_name>] WITH (STATE=ON);
GO
```

Consume audit logs

Consume logs stored in Azure Storage

There are several methods you can use to view blob auditing logs.

- Use the system function `sys.fn_get_audit_file` (T-SQL) to return the audit log data in tabular format. For more information on using this function, see the [sys.fn_get_audit_file documentation](#).
- You can explore audit logs by using a tool such as [Azure Storage Explorer](#). In Azure storage, auditing logs are saved as a collection of blob files within a container that was defined to store the audit logs. For further details about the hierarchy of the storage folder, naming conventions, and log format, see the [Blob Audit Log Format Reference](#).
- For a full list of audit log consumption methods, refer to the [Get started with SQL database auditing](#).

Consume logs stored in Event Hub

To consume audit logs data from Event Hub, you will need to set up a stream to consume events and write them to a target. For more information, see [Azure Event Hubs Documentation](#).

Consume and Analyze logs stored in Azure Monitor logs

If audit logs are written to Azure Monitor logs, they are available in the Log Analytics workspace, where you can run advanced searches on the audit data. As a starting point, navigate to the Log Analytics workspace and under *General* section click *Logs* and enter a simple query, such as: `search "SQLSecurityAuditEvents"` to view the audit logs.

Azure Monitor logs gives you real-time operational insights using integrated search and custom dashboards to readily analyze millions of records across all your workloads and servers. For additional useful information about Azure Monitor logs search language and commands, see [Azure Monitor logs search reference](#).

NOTE

This article was recently updated to use the term Azure Monitor logs instead of Log Analytics. Log data is still stored in a Log Analytics workspace and is still collected and analyzed by the same Log Analytics service. We are updating the terminology to better reflect the role of logs in Azure Monitor. See [Azure Monitor terminology changes](#) for details.

Auditing differences between databases in Azure SQL Database and databases in SQL Server

The key differences between auditing in databases in Azure SQL Database and databases in SQL Server are:

- With the managed instance deployment option in Azure SQL Database, auditing works at the server level and stores `.xel` log files in Azure Blob storage.
- In SQL Server on-premises / virtual machines, audit works at the server level, but stores events on files system/windows event logs.

XEvent auditing in managed instance supports Azure Blob storage targets. File and windows logs are **not supported**.

The key differences in the `CREATE AUDIT` syntax for auditing to Azure Blob storage are:

- A new syntax `TO URL` is provided and enables you to specify URL of the Azure blob Storage container where the `.xel` files are placed.
- A new syntax `TO EXTERNAL MONITOR` is provided to enable Event Hub and Azure Monitor logs targets.
- The syntax `TO FILE` is **not supported** because SQL Database cannot access Windows file shares.
- Shutdown option is **not supported**.
- `queue_delay` of 0 is **not supported**.

Next steps

- For a full list of audit log consumption methods, refer to the [Get started with SQL database auditing](#).
- For more information about Azure programs that support standards compliance, see the [Azure Trust Center](#) where you can find the most current list of SQL Database compliance certifications.

Azure Security Baseline for Azure SQL Database

2/24/2020 • 28 minutes to read • [Edit Online](#)

The Azure Security Baseline for Azure SQL Database contains recommendations that will help you improve the security posture of your deployment.

The baseline for this service is drawn from the [Azure Security Benchmark version 1.0](#), which provides recommendations on how you can secure your cloud solutions on Azure with our best practices guidance.

For more information, see [Azure Security Baselines overview](#).

Network Security

For more information, see [Security Control: Network Security](#).

1.1: Protect resources using Network Security Groups or Azure Firewall on your Virtual Network

Guidance: You can enable Azure Private Link to allow access Azure PaaS Services (for example, SQL Database) and Azure hosted customer/partner services over a Private Endpoint in your virtual network. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. To allow traffic to reach Azure SQL Database, use the SQL service tags to allow outbound traffic through Network Security Groups.

Virtual network rules enable Azure SQL Database to only accept communications that are sent from selected subnets inside a virtual network.

How to set up Private Link for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-private-endpoint-overview#how-to-set-up-private-link-for-azure-sql-database>

How to use virtual network service endpoints and rules for database servers:

<https://docs.microsoft.com/azure/sql-database/sql-database-vnet-service-endpoint-rule-overview>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.2: Monitor and log the configuration and traffic of Vnets, Subnets, and NICs

Guidance: Use Azure Security Center and remediate network protection recommendations for the subnet your Azure SQL Database Server is deployed to. For Azure Virtual Machines (VM) that will be connecting to your Azure SQL Database Server instance, enable network security group (NSG) flow logs for the NSGs protecting those VMs and send logs into a Azure Storage Account for traffic auditing. You may also send NSG flow logs to a Log Analytics workspace and use Traffic Analytics to provide insights into traffic flow in your Azure cloud. Some advantages of Traffic Analytics are the ability to visualize network activity and identify hot spots, identify security threats, understand traffic flow patterns, and pinpoint network misconfigurations.

How to Enable NSG Flow Logs:

<https://docs.microsoft.com/azure/network-watcher/network-watcher-nsg-flow-logging-portal>

Understand Network Security provided by Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-network-recommendations>

How to Enable and use Traffic Analytics:

<https://docs.microsoft.com/azure/network-watcher/traffic-analytics>

Understand Network Security provided by Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-network-recommendations>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.3: Protect critical web applications

Guidance: Not applicable; this recommendation is intended for Azure Apps Service or compute resources hosting web applications.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

1.4: Deny communications with known malicious IP addresses

Guidance: Enable DDoS Protection Standard on the Virtual Networks associated with your SQL Server instances for protections from distributed denial-of-service attacks. Use Azure Security Center Integrated Threat Intelligence to deny communications with known malicious or unused Internet IP addresses.

How to configure DDoS protection:

<https://docs.microsoft.com/azure/virtual-network/manage-ddos-protection>

Understand Azure Security Center Integrated Threat Intelligence:

<https://docs.microsoft.com/azure/security-center/security-center-alerts-data-services>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.5: Record network packets and flow logs

Guidance: For Azure Virtual Machines (VMs) that will be connecting to your Azure SQL Database instance, enable network security group (NSG) flow logs for the NSGs protecting those VMs and send logs into a Azure Storage Account for traffic audit. If required for investigating anomalous activity, enable Network Watcher packet capture.

How to Enable NSG Flow Logs:

<https://docs.microsoft.com/azure/network-watcher/network-watcher-nsg-flow-logging-portal>

How to enable Network Watcher:

<https://docs.microsoft.com/azure/network-watcher/network-watcher-create>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

1.6: Deploy network based intrusion detection/intrusion prevention systems (IDS/IPS)

Guidance: Enabled Advanced Threat Protection (ATP) for Azure SQL Database. Users receive an alert upon suspicious database activities, potential vulnerabilities, and SQL injection attacks, as well as anomalous database access and queries patterns. Advanced Threat Protection also integrates alerts with Azure Security Center.

Understand and using Advanced Threat Protection for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-threat-detection-overview>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.7: Manage traffic to web applications

Guidance: Not applicable; this recommendation is intended for Azure Apps Service or compute resources hosting web applications.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

1.8: Minimize complexity and administrative overhead of network security rules

Guidance: Use virtual network service tags to define network access controls on network security groups or Azure Firewall. You can use service tags in place of specific IP addresses when creating security rules. By specifying the service tag name (e.g., ApiManagement) in the appropriate source or destination field of a rule, you can allow or deny the traffic for the corresponding service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change.

When using service endpoints for Azure SQL Database, outbound to Azure SQL Database Public IP addresses is required: Network Security Groups (NSGs) must be opened to Azure SQL Database IPs to allow connectivity. You can do this by using NSG service tags for Azure SQL Database.

Understand Service Tags with Service Endpoints for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-vnet-service-endpoint-rule-overview#limitations>

Understand and using Service Tags:

<https://docs.microsoft.com/azure/virtual-network/service-tags-overview>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

1.9: Maintain standard security configurations for network devices

Guidance: Define and implement network security configurations for your Azure SQL Database server instances with Azure Policy. You may use the "Microsoft.Sql" namespace to define custom policy definitions, or use any of the built-in policy definitions designed for Azure SQL Database server network protection. An example of an applicable built-in network security policy for Azure SQL Database server would be: "SQL Server should use a virtual network service endpoint"

Use Azure Blueprints to simplify large scale Azure deployments by packaging key environment artifacts, such as Azure Resource Management templates, Role-based access control (RBAC), and policies, in a single blueprint definition. Easily apply the blueprint to new subscriptions and environments, and fine-tune control and management through versioning.

How to configure and manage Azure Policy: <https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

How to create an Azure Blueprint: <https://docs.microsoft.com/azure/governance/blueprints/create-blueprint-portal>

Azure Security Center monitoring: Yes

Responsibility: Customer

1.10: Document traffic configuration rules

Guidance: Use tags for network security groups (NSG) and other resources related to network security and traffic flow. For individual NSG rules, use the "Description" field to specify business need and/or duration (etc.) for any

rules that allow traffic to/from a network.

Use any of the built-in Azure policy definitions related to tagging, such as "Require tag and its value" to ensure that all resources are created with tags and to notify you of existing untagged resources.

You may use Azure PowerShell or Azure CLI to look-up or perform actions on resources based on their tags.

How to create and use tags:

<https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

1.11: Use automated tools to monitor network resource configurations and detect changes

Guidance: Use Azure Activity Log to monitor network resource configurations and detect changes for network resources related to your Azure SQL Database server instances. Create alerts within Azure Monitor that will trigger when changes to critical network resources take place.

How to view and retrieve Azure Activity Log events:

<https://docs.microsoft.com/azure/azure-monitor/platform/activity-log-view>

How to create alerts in Azure Monitor:

<https://docs.microsoft.com/azure/azure-monitor/platform/alerts-activity-log>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Logging and Monitoring

For more information, see [Security Control: Logging and Monitoring](#).

2.1: Use approved time synchronization sources

Guidance: Microsoft maintains time sources for Azure resources. You may update time synchronization for your compute deployments.

How to configure time synchronization for Azure compute resources:

<https://docs.microsoft.com/azure/virtual-machines/windows/time-sync>

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

2.2: Configure central security log management

Guidance: Enable auditing for Azure SQL Database to track database events and write them to an audit log in your Azure Storage Account, Log Analytics workspace, or Event Hubs.

In addition, you can stream Azure SQL diagnostics telemetry into Azure SQL Analytics, a cloud solution that monitors the performance of Azure SQL databases, elastic pools, and managed instances at scale and across multiple subscriptions. It can help you collect and visualize Azure SQL Database performance metrics, and it has built-in intelligence for performance troubleshooting.

How to setup auditing for your Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-auditing>

How to collect platform logs and metrics with Azure Monitor:

<https://docs.microsoft.com/azure/sql-database/sql-database-metrics-diag-logging>

How to stream diagnostics into Azure SQL Analytics:

<https://docs.microsoft.com/azure/sql-database/sql-database-metrics-diag-logging#stream-into-azure-sql-analytics>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.3: Enable audit logging for Azure resources

Guidance: Enable auditing on your Azure SQL Database server instance and choose a storage location for the audit logs (Azure Storage, Log Analytics, or Event Hub).

How to enable auditing for Azure SQL Server:

<https://docs.microsoft.com/azure/sql-database/sql-database-auditing>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.4: Collect security logs from operating systems

Guidance: Not applicable; this benchmark is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

2.5: Configure security log storage retention

Guidance: When storing your Azure SQL Database logs in an Log Analytics Workspace, set log retention period according to your organization's compliance regulations.

How to set log retention parameters:

<https://docs.microsoft.com/azure/azure-monitor/platform/manage-cost-storage#change-the-data-retention-period>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

2.6: Monitor and review Logs

Guidance: Analyze and monitor logs for anomalous behaviors and regularly review results. Use Azure Security Center's Advanced Threat Protection to alert on unusual activity related to your Azure SQL Database instance. Alternatively, configure alerts based on Metric Values or Azure Activity Log entries related to your Azure SQL Database instances.

Understand Advanced Threat Protection and alerting for Azure SQL Server:

<https://docs.microsoft.com/azure/sql-database/sql-database-threat-detection-overview>

How to configure custom alerts for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-insights-alerts-portal?view=azps-1.4.0>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.7: Enable alerts for anomalous activity

Guidance: Use Azure Security Center Advanced Threat Protection for Azure SQL Databases for monitoring and alerting on anomalous activity. Enable Advanced Data Security for your SQL Databases. Advanced Data Security

includes functionality for discovering and classifying sensitive data, surfacing and mitigating potential database vulnerabilities, and detecting anomalous activities that could indicate a threat to your database.

Understand Advanced Threat Protection and alerting for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-threat-detection-overview>

How to enable Advanced Data Security for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-advanced-data-security>

How to manage alerts in Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-managing-and-responding-alerts>

Azure Security Center monitoring: Yes

Responsibility: Customer

2.8: Centralize anti-malware logging

Guidance: Not applicable; for Azure SQL Server, the anti-malware solution is managed by Microsoft on the underlying platform.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

2.9: Enable DNS query logging

Guidance: Not applicable; DNS logging is not applicable to Azure SQL Server.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

2.10: Enable command-line audit logging

Guidance: Not applicable; command-line auditing is not applicable to Azure SQL Server.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Identity and Access Control

For more information, see [Security Control: Identity and Access Control](#).

3.1: Maintain an inventory of administrative accounts

Guidance: Azure Active Directory (AAD) has built-in roles that must be explicitly assigned and are queryable. Use the AAD PowerShell module to perform ad-hoc queries to discover accounts that are members of administrative groups.

How to get a directory role in Azure AD with PowerShell:

<https://docs.microsoft.com/powershell/module/azuread/get-azureaddirectoryrole?view=azureadps-2.0>

How to get members of a directory role in Azure AD with PowerShell:

<https://docs.microsoft.com/powershell/module/azuread/get-azureaddirectoryrolemember?view=azureadps-2.0>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

3.2: Change default passwords where applicable

Guidance: Azure Active Directory does not have the concept of default passwords. When provisioning an Azure SQL Database instance, it is recommended that you choose to integrate authentication with Azure Active Directory.

How to configure and manage Azure Active Directory authentication with Azure SQL:

<https://docs.microsoft.com/azure/sql-database/sql-database-aad-authentication-configure>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.3: Use dedicated administrative accounts

Guidance: Create policies and procedures around the use of dedicated administrative accounts. Use Azure Security Center Identity and Access Management to monitor the number of administrative accounts.

Understand Azure Security Center Identity and Access:

<https://docs.microsoft.com/azure/security-center/security-center-identity-access>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.4: Use single sign-on (SSO) with Azure Active Directory

Guidance: Not applicable; while you can configure Azure Active Directory Authentication to integrate with Azure SQL Server, single sign-on is not supported.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

3.5: Use multi-factor authentication for all Azure Active Directory based access

Guidance: Enable Azure Active Directory (AAD) Multi-Factor Authentication (MFA) and follow Azure Security Center Identity and Access Management recommendations.

How to enable MFA in Azure:

<https://docs.microsoft.com/azure/active-directory/authentication/howto-mfa-getstarted>

How to monitor identity and access within Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-identity-access>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.6: Use dedicated machines (Privileged Access Workstations) for all administrative tasks

Guidance: Use a Privileged Access Workstation (PAW) with Multi-Factor Authentication MFA configured to log into and configure Azure resources.

Learn about Privileged Access Workstations:

<https://docs.microsoft.com/windows-server/identity/securing-privileged-access/privileged-access-workstations>

How to enable MFA in Azure:

<https://docs.microsoft.com/azure/active-directory/authentication/howto-mfa-getstarted>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

3.7: Log and alert on suspicious activity from administrative accounts

Guidance: Use Azure Active Directory security reports for generation of logs and alerts when suspicious or unsafe activity occurs in the environment.

Use Advanced Threat Protection for Azure SQL Database to detect anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases.

How to identify Azure AD users flagged for risky activity: <https://docs.microsoft.com/azure/active-directory/reports-monitoring/concept-user-at-risk>

How to monitor users identity and access activity in Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-identity-access>

Review Advanced Threat Protection and potential alerts: <https://docs.microsoft.com/azure/sql-database/sql-database-threat-detection-overview#advanced-threat-protection-alerts>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.8: Manage Azure resources from only approved locations

Guidance: Use Conditional Access Named Locations to allow Portal and Azure Resource Management access from only specific logical groupings of IP address ranges or countries/regions.

How to configure Named Locations in Azure: <https://docs.microsoft.com/azure/active-directory/reports-monitoring/quickstart-configure-named-locations>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

3.9: Use Azure Active Directory

Guidance: Create an Azure Active Directory (AAD) administrator for your Azure SQL Database server instances.

How to configure and manage Azure Active Directory authentication with Azure SQL:

<https://docs.microsoft.com/azure/sql-database/sql-database-aad-authentication-configure>

How to create and configure an AAD instance:

<https://docs.microsoft.com/azure/active-directory-domain-services/tutorial-create-instance>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.10: Regularly review and reconcile user access

Guidance: Azure Active Directory (AAD) provides logs to help discover stale accounts. In addition, use Azure Identity access reviews to efficiently manage group memberships, access to enterprise applications, and role assignments. Users' access can be reviewed on a regular basis to make sure only the right users have continued access.

How to use Azure Identity Access Reviews:

<https://docs.microsoft.com/azure/active-directory/governance/access-reviews-overview>

Azure Security Center monitoring: Yes

Responsibility: Customer

3.11: Monitor attempts to access deactivated accounts

Guidance: Configure Azure Active Directory (AAD) authentication with Azure SQL and create Diagnostic Settings for Azure Active Directory user accounts, sending the audit logs and sign-in logs to a Log Analytics workspace. Configure desired Alerts within Log Analytics workspace.

How to configure and manage Azure Active Directory authentication with Azure SQL:

<https://docs.microsoft.com/azure/sql-database/sql-database-aad-authentication-configure>

How to integrate Azure Activity Logs into Azure Monitor:

<https://docs.microsoft.com/azure/active-directory/reports-monitoring/howto-integrate-activity-logs-with-log-analytics>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

3.12: Alert on account login behavior deviation

Guidance: Use Azure Active Directory (AAD) Identity Protection and risk detections to configure automated responses to detected suspicious actions related to user identities. Additionally, you can ingest data into Azure Sentinel for further investigation.

How to view Azure AD risk sign-ins:

<https://docs.microsoft.com/azure/active-directory/reports-monitoring/concept-risky-sign-ins>

How to configure and enable Identity Protection risk policies:

<https://docs.microsoft.com/azure/active-directory/identity-protection/howto-identity-protection-configure-risk-policies>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

3.13: Provide Microsoft with access to relevant customer data during support scenarios

Guidance: In support scenarios where Microsoft needs to access customer data, Azure Customer Lockbox provides an interface for customers to review and approve or reject customer data access requests.

Understand Customer Lockbox:

<https://docs.microsoft.com/azure/security/fundamentals/customer-lockbox-overview>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Data Protection

For more information, see [Security Control: Data Protection](#).

4.1: Maintain an inventory of sensitive Information

Guidance: Use tags to assist in tracking Azure resources that store or process sensitive information.

How to create and use tags:

<https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

4.2: Isolate systems storing or processing sensitive information

Guidance: Implement separate subscriptions and/or management groups for development, test, and production. Resources should be separated by Vnet/Subnet, tagged appropriately, and secured within an NSG or Azure Firewall. Resources storing or processing sensitive data should be isolated. Use Private Link; deploy Azure SQL Server inside your Vnet and connect privately using Private Endpoints.

How to create additional Azure subscriptions:

<https://docs.microsoft.com/azure/billing/billing-create-subscription>

How to create Management Groups:

<https://docs.microsoft.com/azure/governance/management-groups/create>

How to create and use Tags:

<https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

How to set up Private Link for Azure SQL Database:

<https://docs.microsoft.com/azure/sql-database/sql-database-private-endpoint-overview#how-to-set-up-private-link-for-azure-sql-database>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

4.3: Monitor and block unauthorized transfer of sensitive information

Guidance: For Azure SQL Databases storing or processing sensitive information, mark the database and related resources as sensitive using Tags. Configure Private Link in conjunction with Network Security Group Service Tags on your Azure SQL Database instances to prevent the exfiltration of sensitive information.

For the underlying platform which is managed by Microsoft, Microsoft treats all customer content as sensitive and goes to great lengths to guard against customer data loss and exposure. To ensure customer data within Azure remains secure, Microsoft has implemented and maintains a suite of robust data protection controls and capabilities.

How to configure Private Link and NSGs to prevent data exfiltration on your Azure SQL Database instances:

<https://docs.microsoft.com/azure/sql-database/sql-database-private-endpoint-overview>

Understand customer data protection in Azure:

<https://docs.microsoft.com/azure/security/fundamentals/protection-customer-data>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

4.4: Encrypt all sensitive information in transit

Guidance: Azure SQL Database secures your data by encrypting data in motion with Transport Layer Security. SQL Server enforces encryption (SSL/TLS) at all times for all connections. This ensures all data is encrypted "in transit" between the client and server irrespective of the setting of Encrypt or TrustServerCertificate in the connection string.

Understand Azure SQL Encryption in Transit:

<https://docs.microsoft.com/azure/sql-database/sql-database-security-overview#information-protection-and-encryption>

Azure Security Center monitoring: Not applicable

Responsibility: Microsoft

4.5: Use an active discovery tool to identify sensitive data

Guidance: Use the Azure SQL Database data discovery and classification feature. Data discovery and classification provides advanced capabilities built into Azure SQL Database for discovering, classifying, labeling & protecting the sensitive data in your databases.

How to use data discovery and classification for Azure SQL Server:

<https://docs.microsoft.com/azure/sql-database/sql-database-data-discovery-and-classification>

Azure Security Center monitoring: Yes

Responsibility: Customer

4.6: Use Azure RBAC to control access to resources

Guidance: Use Azure Active Directory (AAD) for authenticating and controlling access to Azure SQL Database instances.

How to integrate Azure SQL Server with Azure Active Directory for authentication:

<https://docs.microsoft.com/azure/sql-database/sql-database-aad-authentication>

How to control access in Azure SQL Server:

<https://docs.microsoft.com/azure/sql-database/sql-database-control-access>

Azure Security Center monitoring: Yes

Responsibility: Customer

4.7: Use host-based data loss prevention to enforce access control

Guidance: Microsoft manages the underlying infrastructure for Azure SQL Database and has implemented strict controls to prevent the loss or exposure of customer data.

Understand customer data protection in Azure:

<https://docs.microsoft.com/azure/security/fundamentals/protection-customer-data>

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

4.8: Encrypt sensitive information at rest

Guidance: Transparent data encryption (TDE) helps protect Azure SQL Database, Azure SQL managed instance, and Azure Data Warehouse against the threat of malicious offline activity by encrypting data at rest. It performs real-time encryption and decryption of the database, associated backups, and transaction log files at rest without requiring changes to the application. By default, TDE is enabled for all newly deployed Azure SQL databases. The TDE encryption key can be managed by either Microsoft or the customer.

How to manage transparent data encryption and use your own encryption keys:

<https://docs.microsoft.com/azure/sql-database/transparent-data-encryption-azure-sql?tabs=azure-portal#manage-transparent-data-encryption>

Azure Security Center monitoring: Yes

Responsibility: Customer

4.9: Log and alert on changes to critical Azure resources

Guidance: Use Azure Monitor with the Azure Activity Log to create alerts for when changes take place to production instances of Azure SQL Databases and other critical or related resources.

How to create alerts for Azure Activity Log events:

<https://docs.microsoft.com/azure/azure-monitor/platform/alerts-activity-log>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

Vulnerability Management

For more information, see [Security Control: Vulnerability Management](#).

5.1: Run automated vulnerability scanning tools

Guidance: Enable Advanced Data Security for Azure SQL Database and follow recommendations from Azure Security Center on performing vulnerability assessments on your Azure SQL Servers.

How to run vulnerability assessments on your Azure SQL Databases:

<https://docs.microsoft.com/azure/sql-database/sql-vulnerability-assessment>

How to enable Advanced Data Security:

<https://docs.microsoft.com/azure/sql-database/sql-database-advanced-data-security>

How to implement Azure Security Center vulnerability assessment recommendations:

<https://docs.microsoft.com/azure/security-center/security-center-vulnerability-assessment-recommendations>

Azure Security Center monitoring: Yes

Responsibility: Customer

5.2: Deploy automated operating system patch management solution

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

5.3: Deploy automated third-party software patch management solution

Guidance: Not applicable; this benchmark is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

5.4: Compare back-to-back vulnerability scans

Guidance: Enable periodic recurring scans for your Azure SQL Database instances; this will configure a vulnerability assessment to automatically run a scan on your database once per week. A scan result summary will be sent to the email address(es) you provide. Compare the results to verify that vulnerabilities have been remediated.

How to export a vulnerability assessment report in Azure Security Center:

<https://docs.microsoft.com/azure/sql-database/sql-vulnerability-assessment#implementing-vulnerability-assessment>

Azure Security Center monitoring: Yes

Responsibility: Customer

5.5: Use a risk-rating process to prioritize the remediation of discovered vulnerabilities

Guidance: Use the default risk ratings (Secure Score) provided by Azure Security Center.

Understand Azure Security Center Secure Score: <https://docs.microsoft.com/azure/security-center/security-center-secure-score>

Azure Security Center monitoring: Yes

Responsibility: Customer

Inventory and Asset Management

For more information, see [Security Control: Inventory and Asset Management](#).

6.1: Use Azure Asset Discovery

Guidance: Use Azure Resource Graph to query and discover all resources (including Azure SQL Server instances) within your subscription(s). Ensure you have appropriate (read) permissions in your tenant and are able to enumerate all Azure subscriptions as well as resources within your subscriptions.

Although classic Azure resources may be discovered via Resource Graph, it is highly recommended to create and use Azure Resource Manager resources going forward.

How to create queries with Azure Graph: <https://docs.microsoft.com/azure/governance/resource-graph/first-query-portal>

How to view your Azure Subscriptions: <https://docs.microsoft.com/powershell/module/az.accounts/get-azsubscription?view=azps-3.0.0>

Understand Azure RBAC: <https://docs.microsoft.com/azure/role-based-access-control/overview>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.2: Maintain asset metadata

Guidance: Apply tags to Azure resources giving metadata to logically organize them into a taxonomy.

How to create and use Tags: <https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

6.3: Delete unauthorized Azure resources

Guidance: Use tagging, management groups, and separate subscriptions, where appropriate, to organize and track assets. Reconcile inventory on a regular basis and ensure unauthorized resources are deleted from the subscription in a timely manner.

How to create additional Azure subscriptions: <https://docs.microsoft.com/azure/billing/billing-create-subscription>

How to create Management Groups: <https://docs.microsoft.com/azure/governance/management-groups/create>

How to create and use Tags: <https://docs.microsoft.com/azure/azure-resource-manager/resource-group-using-tags>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

6.4: Maintain an inventory of approved Azure resources and software titles

Guidance: Define list of approved Azure resources and approved software for your compute resources

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.5: Monitor for unapproved Azure resources

Guidance: Use Azure Policy to put restrictions on the type of resources that can be created in customer subscription(s) using the following built-in policy definitions:

- Not allowed resource types
- Allowed resource types

Use Azure Resource Graph to query/discover resources within your subscription(s). Ensure that all Azure resources present in the environment are approved.

How to configure and manage Azure Policy: <https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

How to create queries with Azure Graph: <https://docs.microsoft.com/azure/governance/resource-graph/first-query-portal>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.6: Monitor for unapproved software applications within compute resources

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.7: Remove unapproved Azure resources and software applications

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.8: Use only approved applications

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.9: Use only approved Azure services

Guidance: Use Azure Policy to place restrictions on the type of resources that can be created in customer subscription(s) using the following built-in policy definitions:

- Not allowed resource types
- Allowed resource types

Use Azure Resource Graph to query/discover resources within your subscription(s). Ensure that all Azure resources present in the environment are approved.

How to configure and manage Azure Policy: <https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

How to deny a specific resource type with Azure Policy:

<https://docs.microsoft.com/azure/governance/policy/samples/not-allowed-resource-types>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

6.10: Implement approved application list

Guidance: Not applicable; this recommendation is intended for applications running on compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.11: Limit users' ability to interact with Azure Resources Manager via scripts

Guidance: Use Azure Conditional Access to limit users' ability to interact with Azure Resource Manager by configuring "Block access" for the "Microsoft Azure Management" App.

How to configure Conditional Access to block access to Azure Resource Manager:

<https://docs.microsoft.com/azure/role-based-access-control/conditional-access-azure-management>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

6.12: Limit users' ability to execute scripts within compute resources

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

6.13: Physically or logically segregate high risk applications

Guidance: Not applicable; this recommendation is intended for App Service or compute resources hosting desktop or web applications.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Secure Configuration

For more information, see [Security Control: Secure Configuration](#).

7.1: Establish secure configurations for all Azure resources

Guidance: Use Azure Policy or Azure Security Center recommendations for Azure SQL Servers/Databases to maintain security configurations for all Azure Resources.

How to configure and manage Azure Policy:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Azure Security Center monitoring: Yes

Responsibility: Customer

7.2: Establish secure operating system configurations

Guidance: Not applicable; this recommendation intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.3: Maintain secure Azure resource configurations

Guidance: Use Azure policy [deny] and [deploy if not exist] to enforce secure settings across your Azure resources.

How to configure and manage Azure Policy:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Understand Azure Policy Effects:

<https://docs.microsoft.com/azure/governance/policy/concepts/effects>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.4: Maintain secure operating system configurations

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.5: Securely store configuration of Azure resources

Guidance: If using custom Azure policy definitions, use Azure DevOps or Azure Repos to securely store and manage your code.

How to store code in Azure DevOps:

<https://docs.microsoft.com/azure/devops/repos/git/gitworkflow?view=azure-devops>

Azure Repos Documentation:

<https://docs.microsoft.com/azure/devops/repos/index?view=azure-devops>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.6: Securely store custom operating system images

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.7: Deploy system configuration management tools

Guidance: Use Azure Policy aliases in the "Microsoft.SQL" namespace to create custom policies to alert, audit, and enforce system configurations. Additionally, develop a process and pipeline for managing policy exceptions.

How to configure and manage Azure Policy:

<https://docs.microsoft.com/azure/governance/policy/tutorials/create-and-manage>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.8: Deploy system configuration management tools for operating systems

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.9: Implement automated configuration monitoring for Azure services

Guidance: Leverage Azure Security Center to perform baseline scans for your Azure SQL Servers and Databases.

How to remediate recommendations in Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-sql-service-recommendations>

Azure Security Center monitoring: Yes

Responsibility: Customer

7.10: Implement automated configuration monitoring for operating systems

Guidance: Not applicable; this recommendation is intended for compute resources.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

7.11: Manage Azure secrets securely

Guidance: Use Azure Key Vault to store encryption keys for Azure SQL Database Transparent Data Encryption (TDE).

How to protect sensitive data being stored in Azure SQL Server and store the encryption keys in Azure Key Vault:

<https://docs.microsoft.com/azure/sql-database/sql-database-always-encrypted-azure-key-vault>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

7.12: Manage identities securely and automatically

Guidance: Use Managed Identities to provide Azure services with an automatically managed identity in Azure Active Directory (AAD). Managed Identities allows you to authenticate to any service that supports AAD authentication, including Azure Key Vault, without any credentials in your code.

Tutorial: Use a Windows VM system-assigned managed identity to access Azure SQL:

<https://docs.microsoft.com/azure/active-directory/managed-identities-azure-resources/tutorial-windows-vm-access-sql>

How to configure Managed Identities:

<https://docs.microsoft.com/azure/active-directory/managed-identities-azure-resources/qs-configure-portal-windows-vm>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

7.13: Eliminate unintended credential exposure

Guidance: Implement Credential Scanner to identify credentials within your code. Credential Scanner will also encourage moving discovered credentials to more secure locations such as Azure Key Vault.

How to setup Credential Scanner: <https://secdevtools.azurewebsites.net/helpcredscan.html>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

Malware Defense

For more information, see [Security Control: Malware Defense](#).

8.1: Use centrally managed anti-malware software

Guidance: Not applicable; this recommendation is intended for compute resources. Microsoft handles anti-malware for underlying platform.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

8.2: Pre-scan files to be uploaded to non-compute Azure resources

Guidance: Microsoft anti-malware is enabled on the underlying host that supports Azure services (for example, Azure App Service), however it does not run on customer content.

Pre-scan any content being uploaded to non-compute Azure resources, such as App Service, Data Lake Storage, Blob Storage, Azure SQL Server, etc. Microsoft cannot access your data in these instances.

Understand Microsoft Antimalware for Azure Cloud Services and Virtual Machines:

<https://docs.microsoft.com/azure/security/fundamentals/antimalware>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

8.3: Ensure anti-malware software and signatures are updated

Guidance: Not applicable; this recommendation is intended for compute resources. Microsoft handles anti-malware for underlying platform.

Azure Security Center monitoring: Not applicable

Responsibility: Not applicable

Data Recovery

For more information, see [Security Control: Data Recovery](#).

9.1: Ensure regular automated back ups

Guidance: To protect your business from data loss, Azure SQL Database automatically creates full database backups weekly, differential database backups every 12 hours, and transaction log backups every 5 - 10 minutes. The backups are stored in RA-GRS storage for at least 7 days for all service tiers. All service tiers except Basic support configurable backup retention period for point-in-time restore, up to 35 days.

To meet different compliance requirements, you can select different retention periods for weekly, monthly and/or yearly backups. The storage consumption depends on the selected frequency of backups and the retention period(s).

Understand backups and business continuity with Azure SQL Server:

<https://docs.microsoft.com/azure/sql-database/sql-database-business-continuity>

Azure Security Center monitoring: Not applicable

Responsibility: Shared

9.2: Perform complete system backups and backup any customer managed keys

Guidance: Azure SQL Database automatically creates the database backups that are kept between 7 and 35 days, and uses Azure read-access geo-redundant storage (RA-GRS) to ensure that they are preserved even if the data

center is unavailable. These backups are created automatically. If required, enable long-term geo-redundant backups for your Azure SQL Databases.

If using customer-managed keys for Transparent Data Encryption, ensure your keys are being backed up.

Understand backups in Azure SQL Server:

<https://docs.microsoft.com/azure/sql-database/sql-database-automated-backups?tabs=single-database>

How to backup key vault keys in Azure:

<https://docs.microsoft.com/powershell/module/azurerm.keyvault/backup-azurekeyvaultkey?view=azurermps-6.13.0>

Azure Security Center monitoring: Yes

Responsibility: Customer

9.3: Validate all backups including customer managed keys

Guidance: Ensure ability to periodically perform data restoration of content within Azure Backup. If necessary, test restore content to an isolated VLAN. Test restoration of backed up customer-managed keys.

How to restore key vault keys in Azure:

<https://docs.microsoft.com/powershell/module/azurerm.keyvault/restore-azurekeyvaultkey?view=azurermps-6.13.0>

How to recover Azure SQL Database backups using point-in-time restore:

<https://docs.microsoft.com/azure/sql-database/sql-database-recovery-using-backups#point-in-time-restore>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

9.4: Ensure protection of backups and customer managed keys

Guidance: Enable soft delete in Azure Key Vault to protect keys against accidental or malicious deletion.

How to enable soft delete in Key Vault:

<https://docs.microsoft.com/azure/storage/blobs/storage-blob-soft-delete?tabs=azure-portal>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

Incident Response

For more information, see [Security Control: Incident Response](#).

10.1: Create an incident response guide

Guidance: Ensure that there are written incident response plans that defines roles of personnel as well as phases of incident handling/management.

How to configure Workflow Automations within Azure Security Center:

<https://docs.microsoft.com/azure/security-center/security-center-planning-and-operations-guide>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

10.2: Create an incident scoring and prioritization procedure

Guidance: Security Center assigns a severity to alerts, to help you prioritize the order in which you attend to each alert, so that when a resource is compromised, you can get to it right away. The severity is based on how confident Security Center is in the finding or the analytic used to issue the alert as well as the confidence level that there was malicious intent behind the activity that led to the alert. Security alerts in Azure Security Center:<https://docs.microsoft.com/azure/security-center/security-center-alerts-overview>

Azure Security Center monitoring: Yes

Responsibility: Customer

10.3: Test security response procedures

Guidance: Conduct exercises to test your systems' incident response capabilities on a regular cadence. Identify weak points and gaps and revise plan as needed.

You can refer to NIST's publication: Guide to Test, Training, and Exercise Programs for IT Plans and Capabilities:

<https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-84.pdf>

Azure Security Center monitoring: Not applicable

Responsibility: Customer

10.4: Provide security incident contact details and configure alert notifications for security incidents

Guidance: Security incident contact information will be used by Microsoft to contact you if the Microsoft Security Response Center (MSRC) discovers that your data has been accessed by an unlawful or unauthorized party.

How to set the Azure Security Center Security Contact:

<https://docs.microsoft.com/azure/security-center/security-center-provide-security-contact-details>

Azure Security Center monitoring: Yes

Responsibility: Customer

10.5: Incorporate security alerts into your incident response system

Guidance: Export your Azure Security Center alerts and recommendations using the Continuous Export feature. Continuous Export allows you to export alerts and recommendations either manually or in an ongoing, continuous fashion. You may use the Azure Security Center data connector to stream the alerts to Sentinel.

How to configure continuous export:

<https://docs.microsoft.com/azure/security-center/continuous-export>

How to stream alerts into Azure Sentinel:

<https://docs.microsoft.com/azure/sentinel/connect-azure-security-center>

Azure Security Center monitoring: Currently not available

Responsibility: Customer

10.6: Automate the response to security alerts

Guidance: Use the Workflow Automation feature in Azure Security Center to automatically trigger responses via "Logic Apps" on security alerts and recommendations.

How to configure Workflow Automation and Logic Apps:

<https://docs.microsoft.com/azure/security-center/workflow-automation>

Azure Security Center monitoring: Yes

Responsibility: Customer

Penetration Tests and Red Team Exercises

For more information, see [Security Control: Penetration Tests and Red Team Exercises](#).

11.1: Conduct regular penetration testing of your Azure resources and ensure remediation of all critical security findings within 60 days

Guidance: Please follow the Microsoft Rules of Engagement to ensure your Penetration Tests are not in violation of Microsoft policies:

<https://www.microsoft.com/msrc/pentest-rules-of-engagement?rtc=1>.

You can find more information on Microsoft's strategy and execution of Red Teaming and live site penetration testing against Microsoft managed cloud infrastructure, services and applications, here:

<https://gallery.technet.microsoft.com/Cloud-Red-Teaming-b837392e>

Azure Security Center monitoring: Not applicable

Responsibility: Shared

Quickstarts: Azure SQL Database connect and query

11/7/2019 • 4 minutes to read • [Edit Online](#)

The following document includes links to Azure examples showing how to connect and query an Azure SQL database. It also provides some recommendations for Transport Level Security.

Quickstarts

SQL Server Management Studio	This quickstart demonstrates how to use SSMS to connect to an Azure SQL database, and then use Transact-SQL statements to query, insert, update, and delete data in the database.
Azure Data Studio	This quickstart demonstrates how to use Azure Data Studio to connect to an Azure SQL database, and then use Transact-SQL (T-SQL) statements to create the TutorialDB used in Azure Data Studio tutorials.
Azure portal	This quickstart demonstrates how to use the Query editor to connect to a SQL database, and then use Transact-SQL statements to query, insert, update, and delete data in the database.
Visual Studio Code	This quickstart demonstrates how to use Visual Studio Code to connect to an Azure SQL database, and then use Transact-SQL statements to query, insert, update, and delete data in the database.
.NET with Visual Studio	This quickstart demonstrates how to use the .NET framework to create a C# program with Visual Studio to connect to an Azure SQL database and use Transact-SQL statements to query data.
.NET core	This quickstart demonstrates how to use .NET Core on Windows/Linux/macOS to create a C# program to connect to an Azure SQL database and use Transact-SQL statements to query data.
Go	This quickstart demonstrates how to use Go to connect to an Azure SQL database. Transact-SQL statements to query and modify data are also demonstrated.
Java	This quickstart demonstrates how to use Java to connect to an Azure SQL database and then use Transact-SQL statements to query data.
Node.js	This quickstart demonstrates how to use Node.js to create a program to connect to an Azure SQL database and use Transact-SQL statements to query data.

PHP	This quickstart demonstrates how to use PHP to create a program to connect to an Azure SQL database and use Transact-SQL statements to query data.
Python	This quickstart demonstrates how to use Python to connect to an Azure SQL database and use Transact-SQL statements to query data.
Ruby	This quickstart demonstrates how to use Ruby to create a program to connect to an Azure SQL database and use Transact-SQL statements to query data.
R	This quickstart demonstrates how to use R with Azure SQL Database Machine Learning Services to create a program to connect to an Azure SQL database and use Transact-SQL statements to query data.

TLS considerations for SQL Database connectivity

Transport Layer Security (TLS) is used by all drivers that Microsoft supplies or supports for connecting to Azure SQL Database. No special configuration is necessary. For all connections to SQL Server or to Azure SQL Database, we recommend that all applications set the following configurations, or their equivalents:

- **Encrypt = On**
- **TrustServerCertificate = Off**

Some systems use different yet equivalent keywords for those configuration keywords. These configurations ensure that the client driver verifies the identity of the TLS certificate received from the server.

We also recommend that you disable TLS 1.1 and 1.0 on the client if you need to comply with Payment Card Industry - Data Security Standard (PCI-DSS).

Non-Microsoft drivers might not use TLS by default. This can be a factor when connecting to Azure SQL Database. Applications with embedded drivers might not allow you to control these connection settings. We recommend that you examine the security of such drivers and applications before using them on systems that interact with sensitive data.

Libraries

You can use various libraries and frameworks to connect to Azure SQL Database. Check out our [Get started tutorials](#) to quickly get started with programming languages such as C#, Java, Node.js, PHP, and Python. Then build an app by using SQL Server on Linux or Windows or Docker on macOS.

The following table lists connectivity libraries or *drivers* that client applications can use from a variety of languages to connect to and use SQL Server running on-premises or in the cloud. You can use them on Linux, Windows, or Docker and use them to connect to Azure SQL Database and Azure SQL Data Warehouse.

LANGUAGE	PLATFORM	ADDITIONAL RESOURCES	DOWNLOAD	GET STARTED
C#	Windows, Linux, macOS	Microsoft ADO.NET for SQL Server	Download	Get started

LANGUAGE	PLATFORM	ADDITIONAL RESOURCES	DOWNLOAD	GET STARTED
Java	Windows, Linux, macOS	Microsoft JDBC driver for SQL Server	Download	Get started
PHP	Windows, Linux, macOS	PHP SQL driver for SQL Server	Download	Get started
Node.js	Windows, Linux, macOS	Node.js driver for SQL Server	Install	Get started
Python	Windows, Linux, macOS	Python SQL driver	Install choices: * pymssql * pyodbc	Get started
Ruby	Windows, Linux, macOS	Ruby driver for SQL Server	Install	Get started
C++	Windows, Linux, macOS	Microsoft ODBC driver for SQL Server	Download	

The following table lists examples of object-relational mapping (ORM) frameworks and web frameworks that client applications can use with SQL Server running on-premises or in the cloud. You can use the frameworks on Linux, Windows, or Docker and use them to connect to SQL Database and SQL Data Warehouse.

LANGUAGE	PLATFORM	ORM(S)
C#	Windows, Linux, macOS	Entity Framework Entity Framework Core
Java	Windows, Linux, macOS	Hibernate ORM
PHP	Windows, Linux, macOS	Laravel (Eloquent) Doctrine
Node.js	Windows, Linux, macOS	Sequelize ORM
Python	Windows, Linux, macOS	Django
Ruby	Windows, Linux, macOS	Ruby on Rails

Next steps

- For connectivity architecture information, see [Azure SQL Database Connectivity Architecture](#).
- Find [SQL Server drivers](#) that are used to connect from client applications
- Connect to SQL Database:
 - [Connect to SQL Database by using .NET \(C#\)](#)
 - [Connect to SQL Database by using PHP](#)
 - [Connect to SQL Database by using Node.js](#)
 - [Connect to SQL Database by using Java](#)
 - [Connect to SQL Database by using Python](#)

- [Connect to SQL Database by using Ruby](#)
- Retry logic code examples:
 - [Connect resiliently to SQL with ADO.NET](#)
 - [Connect resiliently to SQL with PHP](#)

Quickstart: Use SQL Server Management Studio to connect and query an Azure SQL database

11/25/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, you'll use [SQL Server Management Studio](#) (SSMS) to connect to an Azure SQL database. You'll then run Transact-SQL statements to query, insert, update, and delete data. You can use SSMS to manage any SQL infrastructure, from SQL Server to SQL Database for Microsoft Windows.

Prerequisites

An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

Install the latest SSMS

Before you start, make sure you've installed the latest [SSMS](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the [SQL databases](#) or [SQL managed instances](#) page.

3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Connect to your database

In SMSS, connect to your Azure SQL Database server.

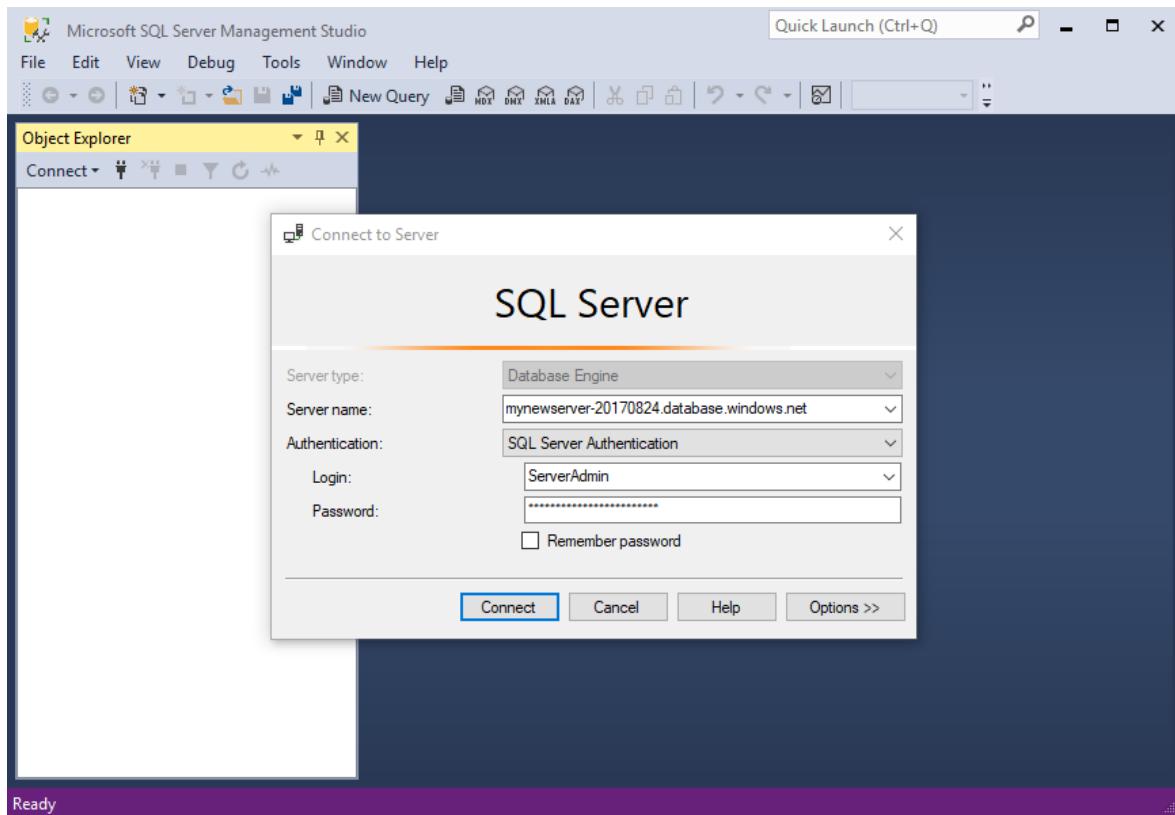
IMPORTANT

An Azure SQL Database server listens on port 1433. To connect to a SQL Database server from behind a corporate firewall, the firewall must have this port open.

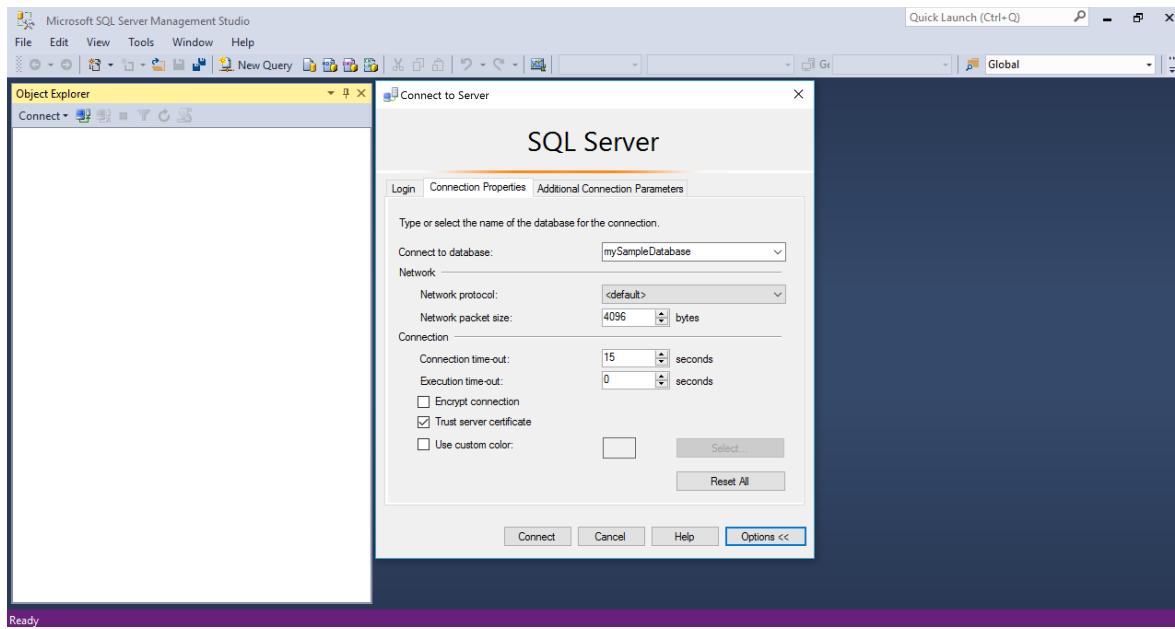
1. Open SSMS. The **Connect to Server** dialog box appears.

2. Enter the following information:

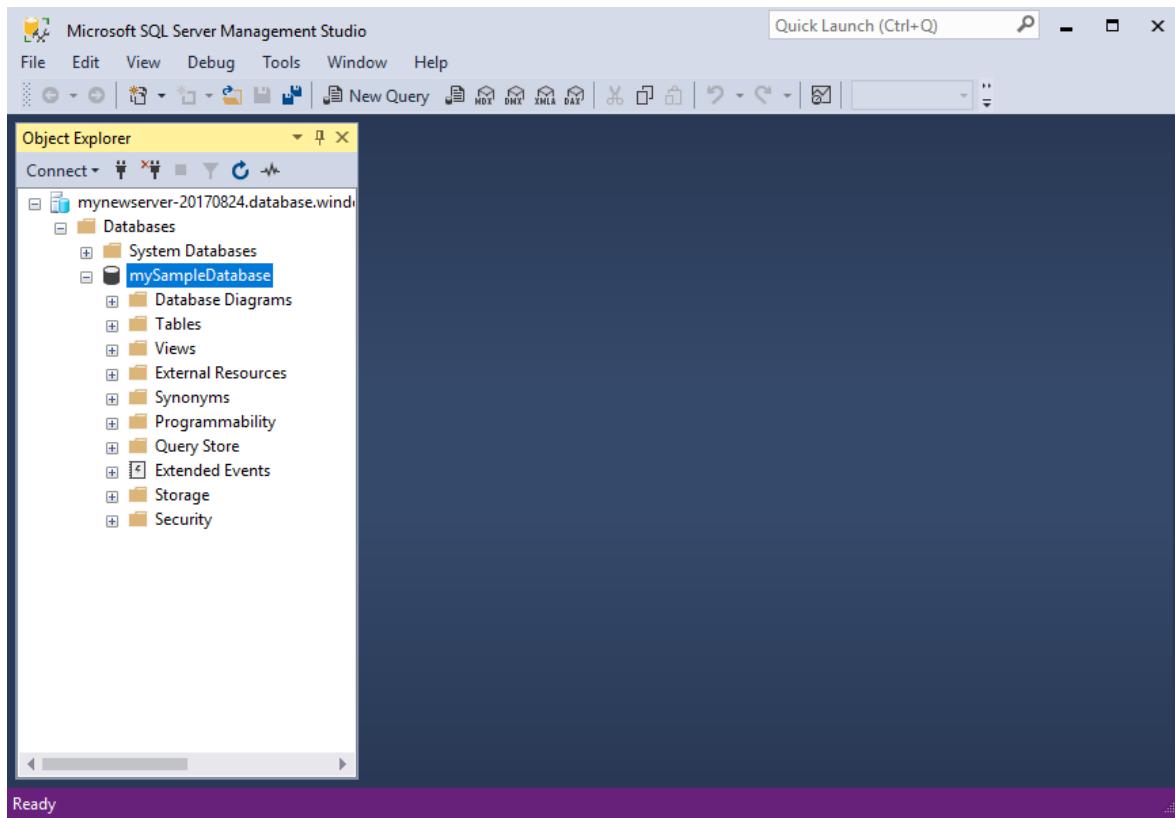
SETTING	SUGGESTED VALUE	DESCRIPTION
Server type	Database engine	Required value.
Server name	The fully qualified server name	Something like: mynewserver20170313.database.windows.net .
Authentication	SQL Server Authentication	This tutorial uses SQL Authentication.
Login	Server admin account user ID	The user ID from the server admin account used to create the server.
Password	Server admin account password	The password from the server admin account used to create the server.



3. Select **Options** in the **Connect to Server** dialog box. In the **Connect to database** drop-down menu, select **mySampleDatabase**. If you leave the drop down to default, the connection is made to **master** database.



4. Select **Connect**. The Object Explorer window opens.
5. To view the database's objects, expand **Databases** and then expand **mySampleDatabase**.



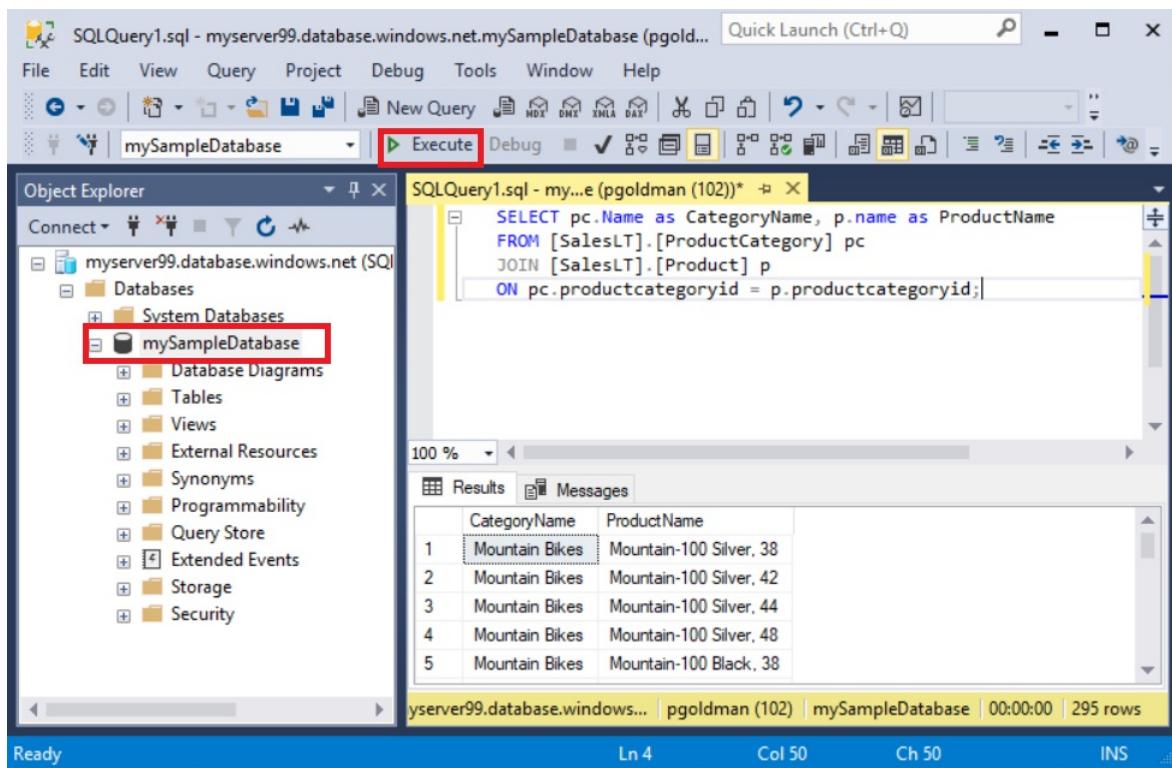
Query data

Run this **SELECT** Transact-SQL code to query for the top 20 products by category.

1. In Object Explorer, right-click **mySampleDatabase** and select **New Query**. A new query window connected to your database opens.
2. In the query window, paste this SQL query.

```
SELECT pc.Name as CategoryName, p.name as ProductName
FROM [SalesLT].[ProductCategory] pc
JOIN [SalesLT].[Product] p
ON pc.productcategoryid = p.productcategoryid;
```

3. On the toolbar, select **Execute** to retrieve data from the **Product** and **ProductCategory** tables.



Insert data

Run this **INSERT** Transact-SQL code to create a new product in the `SalesLT.Product` table.

1. Replace the previous query with this one.

```
INSERT INTO [SalesLT].[Product]
( [Name]
, [ProductNumber]
, [Color]
, [ProductCategoryID]
, [StandardCost]
, [ListPrice]
, [SellStartDate] )
VALUES
('myNewProduct'
,123456789
,'NewColor'
,1
,100
,100
,GETDATE() );
```

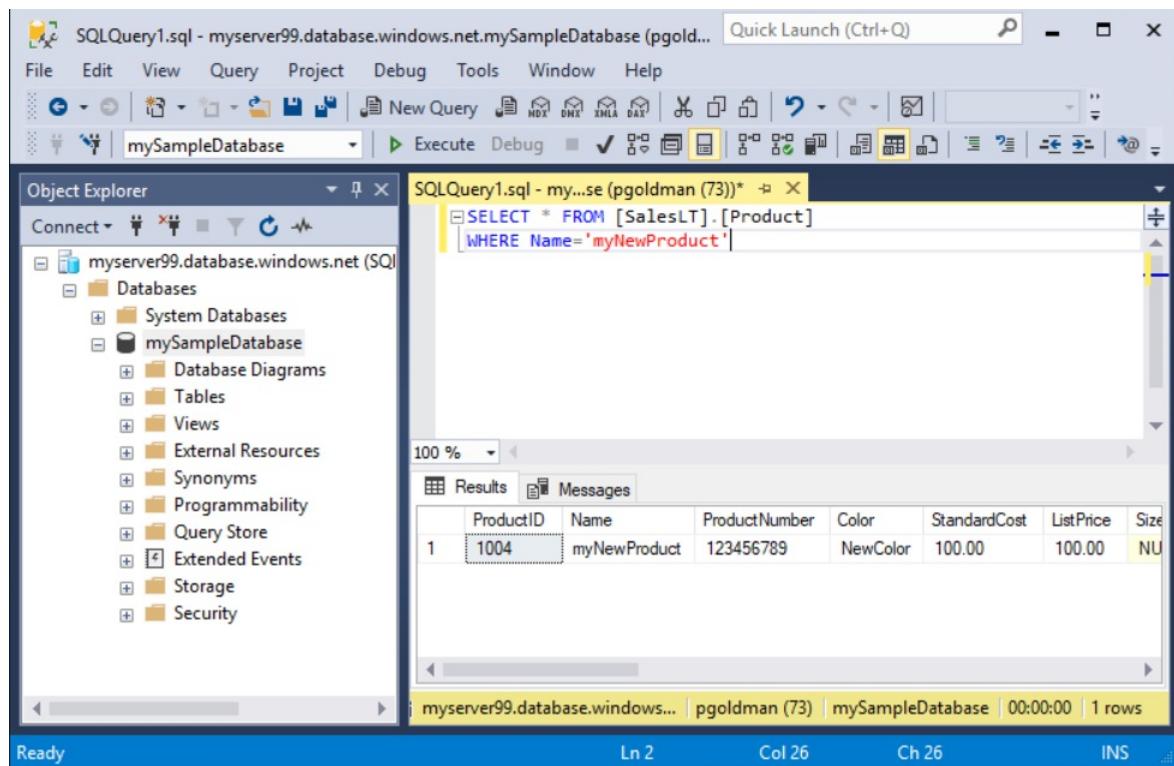
2. Select **Execute** to insert a new row in the `Product` table. The **Messages** pane displays **(1 row affected)**.

View the result

1. Replace the previous query with this one.

```
SELECT * FROM [SalesLT].[Product]
WHERE Name='myNewProduct'
```

2. Select **Execute**. The following result appears.



Update data

Run this **UPDATE** Transact-SQL code to modify your new product.

1. Replace the previous query with this one.

```
UPDATE [SalesLT].[Product]
SET [ListPrice] = 125
WHERE Name = 'myNewProduct';
```

2. Select **Execute** to update the specified row in the **Product** table. The **Messages** pane displays **(1 row affected)**.

Delete data

Run this **DELETE** Transact-SQL code to remove your new product.

1. Replace the previous query with this one.

```
DELETE FROM [SalesLT].[Product]
WHERE Name = 'myNewProduct';
```

2. Select **Execute** to delete the specified row in the **Product** table. The **Messages** pane displays **(1 row affected)**.

Next steps

- For information about SSMS, see [SQL Server Management Studio](#).
- To connect and query using the Azure portal, see [Connect and query with the Azure portal SQL Query editor](#).
- To connect and query using Visual Studio Code, see [Connect and query with Visual Studio Code](#).
- To connect and query using .NET, see [Connect and query with .NET](#).
- To connect and query using PHP, see [Connect and query with PHP](#).

- To connect and query using Node.js, see [Connect and query with Node.js](#).
- To connect and query using Java, see [Connect and query with Java](#).
- To connect and query using Python, see [Connect and query with Python](#).
- To connect and query using Ruby, see [Connect and query with Ruby](#).

Quickstart: Use the Azure portal's SQL query editor to connect and query data

11/7/2019 • 4 minutes to read • [Edit Online](#)

The SQL query editor is an Azure portal browser tool providing an easy way to execute SQL queries on your Azure SQL Database or Azure SQL Data Warehouse. In this quickstart, you'll use the query editor to connect to a SQL database and then run Transact-SQL statements to query, insert, update, and delete data.

Prerequisites

To complete this tutorial, you need:

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE
Create	Portal
	CLI
	PowerShell
Configure	Server-level IP firewall rule

NOTE

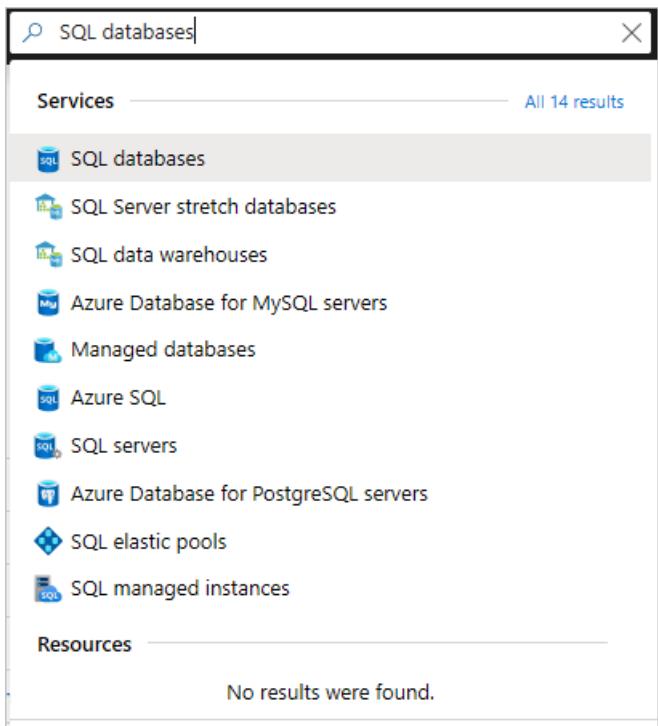
The query editor uses ports 443 and 1443 to communicate. Please ensure you have enabled outbound HTTPS traffic on these ports. You will also need to add your outbound IP address to the server's allowed firewall rules to access your databases and data warehouses.

Sign in the Azure portal

Sign in to the [Azure portal](#).

Connect using SQL authentication

- Go to the Azure portal to connect to a SQL database. Search for and select **SQL databases**.

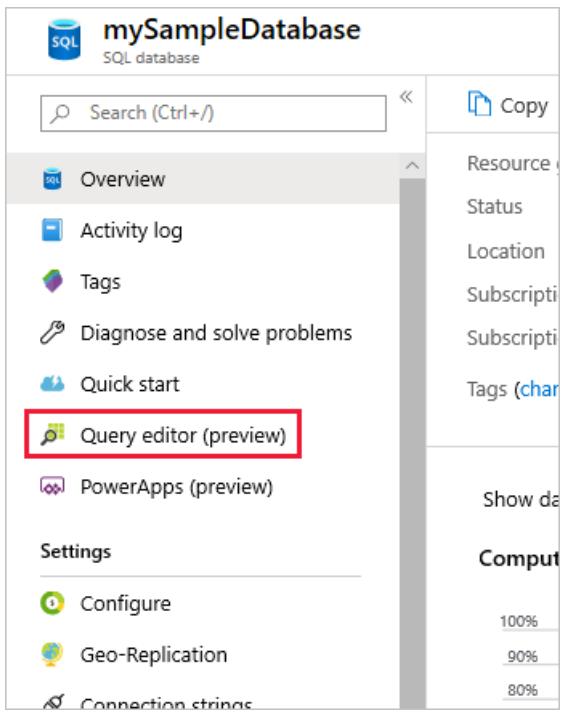


2. Select your SQL database.

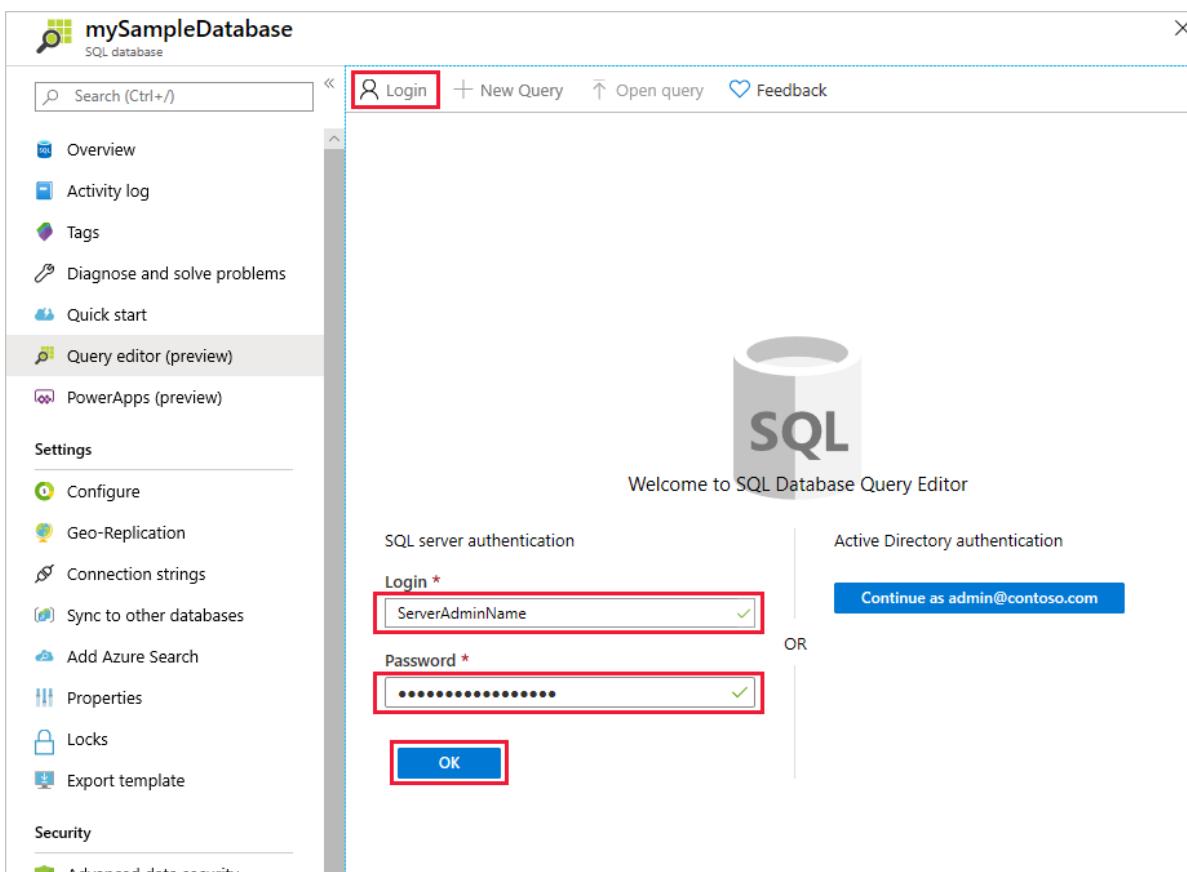
The screenshot shows the Microsoft Azure SQL databases management interface. At the top, there's a navigation bar with 'Home > SQL databases' and a title 'SQL databases' from the Microsoft Azure portal. Below the title are buttons for '+ Add', 'Reservations', 'Edit columns', 'Refresh', and 'Assign tags'. A callout message says 'Try our new Azure SQL resource browser! This experience offers a unified view of all your resources.' Under the heading 'Subscriptions: 2 of 3 selected – Don't see a subscription? Open Directory + Subscriptions', there's a 'Filter by name...' input field and a button '2 subscriptions'. The main area shows a table with '2 items':

<input type="checkbox"/>	Name ↑↓	Status
<input type="checkbox"/>	mySampleDatabase	Online

3. In the **SQL database** menu, select **Query editor (preview)**.



4. In the **Login** page, under the **SQL server authentication** label, enter the **Login ID** and **Password** of the server admin account used to create the database. Then select **OK**.



Connect using Azure Active Directory

Configuring an Azure Active Directory (Azure AD) administrator enables you to use a single identity to sign in to the Azure portal and your SQL database. Follow the steps below to configure an Azure AD admin for your SQL server.

NOTE

- Email accounts (for example, outlook.com, gmail.com, yahoo.com, and so on) aren't yet supported as Azure AD admins. Make sure to choose a user created either natively in the Azure AD, or federated into the Azure AD.
- Azure AD admin sign in doesn't work with accounts that have 2-factor authentication enabled.

1. On the Azure portal menu or from the **Home** page, select **All resources**.
2. Select your SQL server.
3. From the **SQL server** menu, under **Settings**, select **Active Directory admin**.
4. From the SQL server **Active Directory admin** page toolbar, select **Set admin** and choose the user or group as your Azure AD admin.

The screenshot shows the 'mySampleSQLServer - Active Directory admin' page. At the top, there's a search bar and a toolbar with 'Set admin', 'Remove admin', and 'Save' buttons. A callout message says: 'Azure Active Directory authentication allows you to centrally manage identity and access to your Azure SQL Database V12.' Below it, there are two status indicators: 'Active Directory admin ⓘ' (which is greyed out) and 'No Active Directory admin'. The left sidebar lists various management options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Quick start, Failover groups, Manage Backups), Active Directory admin (which is selected and highlighted in grey), SQL databases, and SQL elastic pools.

5. From the **Add admin** page, in the search box, enter a user or group to find, select it as an admin, and then choose the **Select** button.
6. Back in the SQL server **Active Directory admin** page toolbar, select **Save**.
7. In the **SQL server** menu, select **SQL databases**, and then select your SQL database.
8. In the **SQL database** menu, select **Query editor (preview)**. In the **Login** page, under the **Active Directory authentication** label, a message appears saying you have been signed in if you're an Azure AD admin. Then select the **Continue as <your user or group ID>** button.

View data

1. After you're authenticated, paste the following SQL in the query editor to retrieve the top 20 products by category.

```
SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
FROM SalesLT.ProductCategory pc
JOIN SalesLT.Product p
ON pc.productcategoryid = p.productcategoryid;
```

2. On the toolbar, select **Run** and then review the output in the **Results** pane.

The screenshot shows the Microsoft Data Explorer interface. On the left, there's a sidebar for 'mySampleDatabase (ninarn)' with options for Tables, Views, and Stored Procedures. The main area has a 'Query 1' tab with a 'Run' button highlighted by a red box. The query itself is:

```
1 SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
2 FROM SalesLT.ProductCategory pc
3 JOIN SalesLT.Product p
4 ON pc.productcategoryid = p.productcategoryid;
```

Below the query results pane, there are tabs for 'Results' (which is selected) and 'Messages'. The 'Results' tab displays a table of data:

CATEGORYNAME	PRODUCTNAME
Road Frames	HL Road Frame - Black, 58
Road Frames	HL Road Frame - Red, 58
Helmets	Sport-100 Helmet, Red
Helmets	Sport-100 Helmet, Black
Socks	Mountain Bike Socks, M
Socks	Mountain Bike Socks, L
Helmets	Sport-100 Helmet, Blue
Caps	AWC Logo Cap

A green message at the bottom says 'Query succeeded | 5s'.

Insert data

Run the following **INSERT** Transact-SQL statement to add a new product in the `SalesLT.Product` table.

1. Replace the previous query with this one.

```
INSERT INTO [SalesLT].[Product]
( [Name]
, [ProductNumber]
, [Color]
, [ProductCategoryID]
, [StandardCost]
, [ListPrice]
, [SellStartDate]
)
VALUES
('myNewProduct'
,123456789
,'NewColor'
,1
,100
,100
,GETDATE() );
```

2. Select **Run** to insert a new row in the `Product` table. The **Messages** pane displays **Query succeeded: Affected rows: 1.**

Update data

Run the following **UPDATE** Transact-SQL statement to modify your new product.

1. Replace the previous query with this one.

```
UPDATE [SalesLT].[Product]
SET [ListPrice] = 125
WHERE Name = 'myNewProduct';
```

2. Select **Run** to update the specified row in the `Product` table. The **Messages** pane displays **Query succeeded: Affected rows: 1.**

Delete data

Run the following `DELETE` Transact-SQL statement to remove your new product.

1. Replace the previous query with this one:

```
DELETE FROM [SalesLT].[Product]
WHERE Name = 'myNewProduct';
```

2. Select **Run** to delete the specified row in the `Product` table. The **Messages** pane displays **Query succeeded: Affected rows: 1.**

Query editor considerations

There are a few things to know when working with the query editor.

- The query editor uses ports 443 and 1443 to communicate. Please ensure you have enabled outbound HTTPS traffic on these ports. You will also need to add your outbound IP address to the server's allowed firewall rules to access your databases and data warehouses.
- Pressing F5 refreshes the query editor page and any query being worked on is lost.
- Query editor doesn't support connecting to the `master` database.
- There's a 5-minute timeout for query execution.
- The query editor only supports cylindrical projection for geography data types.
- There's no support for IntelliSense for database tables and views. However, the editor does support autocomplete on names that have already been typed.

Next steps

To learn more about the Transact-SQL supported in Azure SQL databases, see [Resolving Transact-SQL differences during migration to SQL Database](#).

Quickstart: Use Visual Studio Code to connect and query an Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

Visual Studio Code is a graphical code editor for Linux, macOS, and Windows. It supports extensions, including the [mssql extension](#) for querying Microsoft SQL Server, Azure SQL Database, and SQL Data Warehouse. In this quickstart, you'll use Visual Studio Code to connect to an Azure SQL database and then run Transact-SQL statements to query, insert, update, and delete data.

Prerequisites

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

Install Visual Studio Code

Make sure you have installed the latest [Visual Studio Code](#) and loaded the [mssql extension](#). For guidance on installing the mssql extension, see [Install VS Code](#) and [mssql for Visual Studio Code](#).

Configure Visual Studio Code

Mac OS

For macOS, you need to install OpenSSL, which is a prerequisite for .NET Core that mssql extension uses. Open

your terminal and enter the following commands to install **brew** and **OpenSSL**.

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew update
brew install openssl
mkdir -p /usr/local/lib
ln -s /usr/local/opt/openssl/lib/libcrypto.1.0.0.dylib /usr/local/lib/
ln -s /usr/local/opt/openssl/lib/libssl.1.0.0.dylib /usr/local/lib/
```

Linux (Ubuntu)

No special configuration needed.

Windows

No special configuration needed.

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Set language mode to SQL

In Visual Studio Code, set the language mode to **SQL** to enable mssql commands and T-SQL IntelliSense.

1. Open a new Visual Studio Code window.
2. Press **Ctrl+N**. A new plain text file opens.
3. Select **Plain Text** in the status bar's lower right-hand corner.
4. In the **Select language mode** drop-down menu that opens, select **SQL**.

Connect to your database

Use Visual Studio Code to establish a connection to your Azure SQL Database server.

IMPORTANT

Before continuing, make sure that you have your server and sign in information ready. Once you begin entering the connection profile information, if you change your focus from Visual Studio Code, you have to restart creating the profile.

1. In Visual Studio Code, press **Ctrl+Shift+P** (or **F1**) to open the Command Palette.
2. Select **MS SQL:Connect** and choose **Enter**.
3. Select **Create Connection Profile**.
4. Follow the prompts to specify the new profile's connection properties. After specifying each value, choose **Enter** to continue.

PROPERTY	SUGGESTED VALUE	DESCRIPTION
Server name	The fully qualified server name	Something like: mynewserver20170313.database.windows.net.
Database name	mySampleDatabase	The database to connect to.
Authentication	SQL Login	This tutorial uses SQL Authentication.
User name	User name	The user name of the server admin account used to create the server.
Password (SQL Login)	Password	The password of the server admin account used to create the server.
Save Password?	Yes or No	Select Yes if you do not want to enter the password each time.
Enter a name for this profile	A profile name, such as mySampleProfile	A saved profile speeds your connection on subsequent logins.

If successful, a notification appears saying your profile is created and connected.

Query data

Run the following **SELECT** Transact-SQL statement to query for the top 20 products by category.

1. In the editor window, paste the following SQL query.

```
SELECT pc.Name as CategoryName, p.name as ProductName  
FROM [SalesLT].[ProductCategory] pc  
JOIN [SalesLT].[Product] p  
ON pc.productcategoryid = p.productcategoryid;
```

2. Press **Ctrl+Shift+E** to run the query and display results from the `Product` and `ProductCategory` tables.

The screenshot shows a Microsoft Data Studio interface. On the left, there are several icons: a file icon with a '1' (indicating one recent file), a magnifying glass, a wrench, and a refresh symbol. The main area has tabs for 'Untitled-1' (selected) and 'Results: Untitled-1'. The code editor contains the following SQL query:

```
1 SELECT pc.Name as CategoryName, p.name as ProductName
2 FROM [SalesLT].[ProductCategory] pc
3 JOIN [SalesLT].[Product] p
4 ON pc.productcategoryid =
p.productcategoryid;
```

The results table shows the output of the query:

	CategoryName	ProductName
1	Mountain Bikes	Mountain-100...
2	Mountain Bikes	Mountain-100...
3	Mountain Bikes	Mountain-100...
4	Mountain Bikes	Mountain-100...
5	Mountain Bikes	Mountain-100...
6	Mountain Bikes	Mountain-100...
7	Mountain Bikes	Mountain-100...

Below the results, a message bar indicates '(295 rows affected)' and 'Total execution time: 00:00:00.367'.

Insert data

Run the following [INSERT](#) Transact-SQL statement to add a new product into the `SalesLT.Product` table.

1. Replace the previous query with this one.

```
INSERT INTO [SalesLT].[Product]
( [Name]
, [ProductNumber]
, [Color]
, [ProductCategoryID]
, [StandardCost]
, [ListPrice]
, [SellStartDate]
)
VALUES
('myNewProduct'
,123456789
,'NewColor'
,1
,100
,100
,GETDATE() );
```

2. Press **Ctrl+Shift+E** to insert a new row in the `Product` table.

Update data

Run the following [UPDATE](#) Transact-SQL statement to update the added product.

1. Replace the previous query with this one:

```
UPDATE [SalesLT].[Product]
SET [ListPrice] = 125
WHERE Name = 'myNewProduct';
```

2. Press **Ctrl+Shift+E** to update the specified row in the `Product` table.

Delete data

Run the following [DELETE](#) Transact-SQL statement to remove the new product.

1. Replace the previous query with this one:

```
DELETE FROM [SalesLT].[Product]
WHERE Name = 'myNewProduct';
```

2. Press **Ctrl+Shift+E** to delete the specified row in the `Product` table.

Next steps

- To connect and query using SQL Server Management Studio, see [Quickstart: Use SQL Server Management Studio to connect to an Azure SQL Database and query data](#).
- To connect and query using the Azure portal, see [Quickstart: Use the SQL Query editor in the Azure portal to connect and query data](#).
- For an MSDN magazine article on using Visual Studio Code, see [Create a database IDE with MSSQL extension blog post](#).

Quickstart: Use .NET and C# in Visual Studio to connect to and query an Azure SQL database

12/31/2019 • 3 minutes to read • [Edit Online](#)

This quickstart shows how to use the [.NET framework](#) and C# code in Visual Studio to query an Azure SQL database with Transact-SQL statements.

Prerequisites

To complete this quickstart, you need:

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:
 - Create
 - [Portal](#)
 - [CLI](#)
 - [PowerShell](#)
 - Configure
 - Server-level IP firewall rule
 - [Connectivity from a VM](#)
 - [Connectivity from on-site](#)
 - Load data
 - Adventure Works loaded per quickstart
 - [Restore Wide World Importers](#)
 - Restore or import Adventure Works from [BACPAC](#) file from [GitHub](#)

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

- [Visual Studio 2019](#) Community, Professional, or Enterprise edition.

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.

3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create code to query the SQL database

1. In Visual Studio, select **File > New > Project**.
2. In the **New Project** dialog, select **Visual C#**, and then select **Console App (.NET Framework)**.
3. Enter *sqltest* for the project name, and then select **OK**. The new project is created.
4. Select **Project > Manage NuGet Packages**.
5. In **NuGet Package Manager**, select the **Browse** tab, then search for and select **System.Data.SqlClient**.
6. On the **System.Data.SqlClient** page, select **Install**.
 - If prompted, select **OK** to continue with the installation.
 - If a **License Acceptance** window appears, select **I Accept**.
7. When the install completes, you can close **NuGet Package Manager**.
8. In the code editor, replace the **Program.cs** contents with the following code. Replace your values for `<server>` , `<username>` , `<password>` , and `<database>` .

IMPORTANT

The code in this example uses the sample AdventureWorksLT data, which you can choose as source when creating your database. If your database has different data, use tables from your own database in the SELECT query.

```

using System;
using System.Data.SqlClient;
using System.Text;

namespace sqltest
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
                builder.DataSource = "<server>.database.windows.net";
                builder.UserID = "<username>";
                builder.Password = "<password>";
                builder.InitialCatalog = "<database>";

                using (SqlConnection connection = new SqlConnection(builder.ConnectionString))
                {
                    Console.WriteLine("\nQuery data example:");
                    Console.WriteLine("=====\n");

                    StringBuilder sb = new StringBuilder();
                    sb.Append("SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName ");
                    sb.Append("FROM [SalesLT].[ProductCategory] pc ");
                    sb.Append("JOIN [SalesLT].[Product] p ");
                    sb.Append("ON pc.productcategoryid = p.productcategoryid");
                    String sql = sb.ToString();

                    using (SqlCommand command = new SqlCommand(sql, connection))
                    {
                        connection.Open();
                        using (SqlDataReader reader = command.ExecuteReader())
                        {
                            while (reader.Read())
                            {
                                Console.WriteLine("{0} {1}", reader.GetString(0), reader.GetString(1));
                            }
                        }
                    }
                }
            catch (SqlException e)
            {
                Console.WriteLine(e.ToString());
            }
            Console.ReadLine();
        }
    }
}

```

Run the code

1. To run the app, select **Debug > Start Debugging**, or select **Start** on the toolbar, or press **F5**.
2. Verify that the top 20 Category/Product rows from your database are returned, and then close the app window.

Next steps

- Learn how to [connect and query an Azure SQL database using .NET Core](#) on Windows/Linux/macOS.
- Learn about [Getting started with .NET Core on Windows/Linux/macOS using the command line](#).
- Learn how to [Design your first Azure SQL database using SSMS](#) or [Design your first Azure SQL database using .NET](#).

- For more information about .NET, see [.NET documentation](#).
- Retry logic example: [Connect resiliently to SQL with ADO.NET](#).

Quickstart: Use .NET Core (C#) to query an Azure SQL database

11/7/2019 • 4 minutes to read • [Edit Online](#)

In this quickstart, you'll use .NET Core and C# code to connect to an Azure SQL database. You'll then run a Transact-SQL statement to query data.

TIP

The following Microsoft Learn module helps you learn for free how to [Develop and configure an ASP.NET application that queries an Azure SQL Database](#)

Prerequisites

For this tutorial, you need:

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

- [.NET Core for your operating system](#) installed.

NOTE

This quickstart uses the *mySampleDatabase* database. If you want to use a different database, you will need to change the database references and modify the `SELECT` query in the C# code.

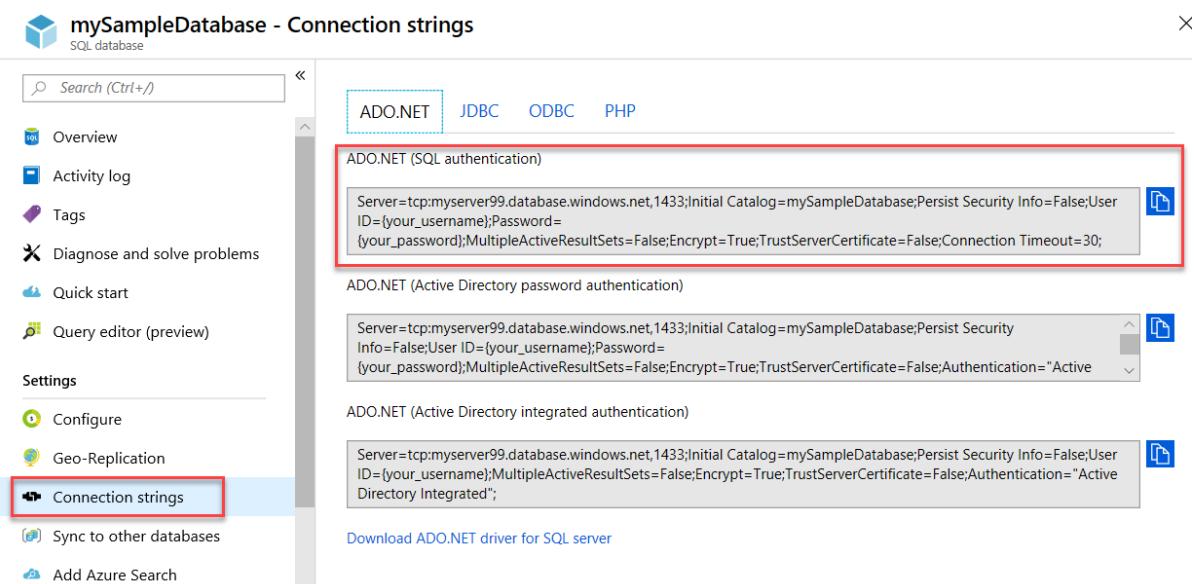
Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Get ADO.NET connection information (optional)

1. Navigate to the **mySampleDatabase** page and, under **Settings**, select **Connection strings**.
2. Review the complete **ADO.NET** connection string.



3. Copy the **ADO.NET** connection string if you intend to use it.

Create a new .NET Core project

1. Open a command prompt and create a folder named **sqltest**. Navigate to this folder and run this command.

```
dotnet new console
```

This command creates new app project files, including an initial C# code file (**Program.cs**), an XML configuration file (**sqltest.csproj**), and needed binaries.

2. In a text editor, open **sqltest.csproj** and paste the following XML between the `<Project>` tags. This XML adds `System.Data.SqlClient` as a dependency.

```
<ItemGroup>
  <PackageReference Include="System.Data.SqlClient" Version="4.6.0" />
</ItemGroup>
```

Insert code to query SQL database

1. In a text editor, open **Program.cs**.
2. Replace the contents with the following code and add the appropriate values for your server, database, username, and password.

NOTE

To use an ADO.NET connection string, replace the 4 lines in the code setting the server, database, username, and password with the line below. In the string, set your username and password.

```
builder.ConnectionString = "<your_ado_net_connection_string>";
```

```

using System;
using System.Data.SqlClient;
using System.Text;

namespace sqltest
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();

                builder.DataSource = "<your_server.database.windows.net>";
                builder.UserID = "<your_username>";
                builder.Password = "<your_password>";
                builder.InitialCatalog = "<your_database>";

                using (SqlConnection connection = new SqlConnection(builder.ConnectionString))
                {
                    Console.WriteLine("\nQuery data example:");
                    Console.WriteLine("=====\n");

                    connection.Open();
                    StringBuilder sb = new StringBuilder();
                    sb.Append("SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName ");
                    sb.Append("FROM [SalesLT].[ProductCategory] pc ");
                    sb.Append("JOIN [SalesLT].[Product] p ");
                    sb.Append("ON pc.productcategoryid = p.productcategoryid");
                    String sql = sb.ToString();

                    using (SqlCommand command = new SqlCommand(sql, connection))
                    {
                        using (SqlDataReader reader = command.ExecuteReader())
                        {
                            while (reader.Read())
                            {
                                Console.WriteLine("{0} {1}", reader.GetString(0), reader.GetString(1));
                            }
                        }
                    }
                }
                catch (SqlException e)
                {
                    Console.WriteLine(e.ToString());
                }
                Console.WriteLine("\nDone. Press enter.");
                Console.ReadLine();
            }
        }
    }
}

```

Run the code

- At the prompt, run the following commands.

```

dotnet restore
dotnet run

```

- Verify that the top 20 rows are returned.

```
Query data example:
```

```
=====
Road Frames HL Road Frame - Black, 58
Road Frames HL Road Frame - Red, 58
Helmets Sport-100 Helmet, Red
Helmets Sport-100 Helmet, Black
Socks Mountain Bike Socks, M
Socks Mountain Bike Socks, L
Helmets Sport-100 Helmet, Blue
Caps AWC Logo Cap
Jerseys Long-Sleeve Logo Jersey, S
Jerseys Long-Sleeve Logo Jersey, M
Jerseys Long-Sleeve Logo Jersey, L
Jerseys Long-Sleeve Logo Jersey, XL
Road Frames HL Road Frame - Red, 62
Road Frames HL Road Frame - Red, 44
Road Frames HL Road Frame - Red, 48
Road Frames HL Road Frame - Red, 52
Road Frames HL Road Frame - Red, 56
Road Frames LL Road Frame - Black, 58
Road Frames LL Road Frame - Black, 60
Road Frames LL Road Frame - Black, 62
```

```
Done. Press enter.
```

3. Choose **Enter** to close the application window.

Next steps

- [Getting started with .NET Core on Windows/Linux/macOS using the command line.](#)
- Learn how to connect and query an Azure SQL database using the .NET Framework and Visual Studio.
- Learn how to Design your first Azure SQL database using SSMS or [Design an Azure SQL database and connect with C# and ADO.NET](#).
- For more information about .NET, see [.NET documentation](#).

Connect to Azure SQL Database with Azure Multi-Factor Authentication

11/6/2019 • 8 minutes to read • [Edit Online](#)

This article provides a C# program that connects to Azure SQL Database. The program uses interactive mode authentication, which supports [Azure Multi-Factor Authentication](#).

For more information about Multi-Factor Authentication support for SQL tools, see [Azure Active Directory support in SQL Server Data Tools \(SSDT\)](#).

Multi-Factor Authentication for Azure SQL Database

Starting in .NET Framework version 4.7.2, the enum `Sq1AuthenticationMethod` has a new value:

`ActiveDirectoryInteractive`. In a client C# program, the enum value directs the system to use the Azure Active Directory (Azure AD) interactive mode that supports Multi-Factor Authentication to connect to an Azure SQL database. The user who runs the program sees the following dialog boxes:

- A dialog box that displays an Azure AD user name and asks for the user's password.

If the user's domain is federated with Azure AD, this dialog box doesn't appear, because no password is needed.

If the Azure AD policy imposes Multi-Factor Authentication on the user, the next two dialog boxes are displayed.

- The first time a user goes through Multi-Factor Authentication, the system displays a dialog box that asks for a mobile phone number to send text messages to. Each message provides the *verification code* that the user must enter in the next dialog box.
- A dialog box that asks for a Multi-Factor Authentication verification code, which the system has sent to a mobile phone.

For information about how to configure Azure AD to require Multi-Factor Authentication, see [Getting started with Azure Multi-Factor Authentication in the cloud](#).

For screenshots of these dialog boxes, see [Configure multi-factor authentication for SQL Server Management Studio and Azure AD](#).

TIP

You can search .NET Framework APIs with the [.NET API Browser tool page](#).

You can also search directly with the `optional ?term=<search value>` parameter.

Configure your C# application in the Azure portal

Before you begin, you should have an [Azure SQL Database server](#) created and available.

Register your app and set permissions

To use Azure AD authentication, your C# program has to register as an Azure AD application. To register an app, you need to be either an Azure AD admin or a user assigned the Azure AD *Application Developer* role. For more information about assigning roles, see [Assign administrator and non-administrator roles to users with Azure](#)

Active Directory.

Completing an app registration generates and displays an **application ID**. Your program has to include this ID to connect.

To register and set necessary permissions for your application:

1. In the Azure portal, select **Azure Active Directory > App registrations > New registration**.

The screenshot shows the 'Register an application' form. At the top, there's a warning message: '⚠ If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide](#)'.

Name: myCSharpApp

Supported account types:

- Accounts in this organizational directory only (Microsoft only - Single tenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

Redirect URI (optional):

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. <https://myapp.com/auth>

By proceeding, you agree to the Microsoft Platform Policies

Register

After the app registration is created, the **application ID** value is generated and displayed.

The screenshot shows the 'Overview' page for the 'myCSharpApp' registration. On the left, there's a sidebar with links: Search (Ctrl+/, Overview, Quickstart, Manage, Branding, Authentication. The main area shows the app's details:

- Display name : myCSharpApp
- Application (client) ID : 4e38b60d-76e4-45f5-8f1d-a7b523cd31b5 (highlighted with a red box)
- Directory (tenant) ID : 72f988bf-86f1-41af-91ab-2d7cd011db47
- Object ID : 55572ae4-d1b7-4112-9e77-965b552e1dc8

2. Select **API permissions > Add a permission**.

Home > Microsoft - App registrations > myCSharpApp - API permissions

myCSharpApp - API permissions

Search (Ctrl+ /)

Overview Quickstart

Manage

Branding Authentication Certificates & secrets API permissions Expose an API Owners Roles and administrators (Previous) Manifest

Support + Troubleshooting Troubleshooting New support request

API permissions

Applications are authorized to call APIs with all the permissions the application needs.

Add a permission

API / PERMISSIONS NAME

▼ Microsoft Graph (1)

User.Read

These are the permissions that this application has been granted. You can manage permissions dynamically through code.

Grant consent

These permissions have been granted for your application, you should consider adding more.

Grant admin consent for Microsoft

3. Select **APIs my organization uses** > type **Azure SQL Database** into the search > and select **Azure SQL Database**.

Request API permissions

Select an API

[Microsoft APIs](#) [APIs my organization uses](#) [My APIs](#)

Apps in your directory that expose APIs are shown below

azure sql database

NAME
Azure SQL Database

4. Select **Delegated permissions** > **user_impersonation** > **Add permissions**.

Request API permissions

◀ All APIs

Azure SQL Database
https://database.windows.net/

What type of permissions does your application require?

Delegated permissions
Your application needs to access the API as the signed-in user.

Application permissions
Your application runs as a background service or daemon without a signed-in user.

Select permissions

Type to search

PERMISSION ADMIN CONSENT REQUIRED

<input checked="" type="checkbox"/> user_impersonation	Access Azure SQL DB and Data Warehouse	
--	--	--

expand all

Set an Azure AD admin for your SQL Database server

For your C# program to run, an Azure SQL server admin needs to assign an Azure AD admin for your SQL Database server.

On the **SQL Server** page, select **Active Directory admin > Set admin**.

For more information about Azure AD admins and users for Azure SQL Database, see the screenshots in [Configure and manage Azure Active Directory authentication with SQL Database](#).

Add a non-admin user to a specific database (optional)

An Azure AD admin for a SQL Database server can run the C# example program. An Azure AD user can run the program if they are in the database. An Azure AD SQL admin or an Azure AD user who exists already in the database and has the `ALTER ANY USER` permission on the database can add a user.

You can add a user to the database with the SQL `Create User` command. An example is

```
CREATE USER [<username>] FROM EXTERNAL PROVIDER .
```

For more information, see [Use Azure Active Directory Authentication for authentication with SQL Database, Managed Instance, or SQL Data Warehouse](#).

New authentication enum value

The C# example relies on the `System.Data.SqlClient` namespace. Of special interest for Multi-Factor Authentication is the enum `SqlAuthenticationMethod`, which has the following values:

- `SqlAuthenticationMethod.ActiveDirectoryInteractive`

Use this value with an Azure AD user name to implement Multi-Factor Authentication. This value is the focus of the present article. It produces an interactive experience by displaying dialog boxes for the user password, and then for Multi-Factor Authentication validation if Multi-Factor Authentication is imposed on this user. This value is available starting with .NET Framework version 4.7.2.

- `SqlAuthenticationMethod.ActiveDirectoryIntegrated`

Use this value for a *federated* account. For a federated account, the user name is known to the Windows domain. This authentication method doesn't support Multi-Factor Authentication.

- `SqlAuthenticationMethod.ActiveDirectoryPassword`

Use this value for authentication that requires an Azure AD user name and password. Azure SQL Database

does the authentication. This method doesn't support Multi-Factor Authentication.

Set C# parameter values from the Azure portal

For the C# program to successfully run, you need to assign proper values to static fields. Shown here are fields with example values. Also shown are the Azure portal locations where you can obtain the needed values.

STATIC FIELD NAME	EXAMPLE VALUE	WHERE IN AZURE PORTAL
Az_SQLDB_svrName	"my-sqldb-srv.database.windows.net"	SQL servers > Filter by name
AzureAD_UserID	"auser@abc.onmicrosoft.com"	Azure Active Directory > User > New guest user
Initial_DatabaseName	"myDatabase"	SQL servers > SQL databases
ClientApplicationID	"a94f9c62-97fe-4d19-b06d-111111111111"	Azure Active Directory > App registrations > Search by name > Application ID
RedirectUri	new Uri("https://mywebserver.com/")	Azure Active Directory > App registrations > Search by name > [Your-App-registration] > Settings > RedirectURIs For this article, any valid value is fine for RedirectUri, because it isn't used here.

Verify with SQL Server Management Studio

Before you run the C# program, it's a good idea to check that your setup and configurations are correct in SQL Server Management Studio (SSMS). Any C# program failure can then be narrowed to source code.

Verify SQL Database firewall IP addresses

Run SSMS from the same computer, in the same building, where you plan to run the C# program. For this test, any **Authentication** mode is OK. If there's any indication that the database server firewall isn't accepting your IP address, see [Azure SQL Database server-level and database-level firewall rules](#) for help.

Verify Azure Active Directory Multi-Factor Authentication

Run SSMS again, this time with **Authentication** set to **Active Directory - Universal with MFA support**. This option requires SSMS version 17.5 or later.

For more information, see [Configure Multi-Factor Authentication for SSMS and Azure AD](#).

NOTE

If you are a guest user in the database, you also need to provide the Azure AD domain name for the database: Select **Options > AD domain name or tenant ID**. To find the domain name in the Azure portal, select **Azure Active Directory > Custom domain names**. In the C# example program, providing a domain name is not necessary.

C# code example

The example C# program relies on the [Microsoft.IdentityModel.Clients.ActiveDirectory](#) DLL assembly.

To install this package, in Visual Studio, select **Project > Manage NuGet Packages**. Search for and install **Microsoft.IdentityModel.Clients.ActiveDirectory**.

This is an example of C# source code.

```
using System;

// Reference to Azure AD authentication assembly
using Microsoft.IdentityModel.Clients.ActiveDirectory;

using DA = System.Data;
using SC = System.Data.SqlClient;
using AD = Microsoft.IdentityModel.Clients.ActiveDirectory;
using TX = System.Text;
using TT = System.Threading.Tasks;

namespace ADInteractive
{
    class Program
    {
        // ASSIGN YOUR VALUES TO THESE STATIC FIELDS !!
        static public string Az_SQLDB_svrName = "<Your SQL DB server>";
        static public string AzureAD_UserID = "<Your User ID>";
        static public string Initial_DatabaseName = "<Your Database>";
        // Some scenarios do not need values for the following two fields:
        static public readonly string ClientApplicationID = "<Your App ID>";
        static public readonly Uri RedirectUri = new Uri("<Your URI>");

        public static void Main(string[] args)
        {
            var provider = new ActiveDirectoryAuthProvider();

            SC.SqlAuthenticationProvider.SetProvider(
                SC.SqlAuthenticationMethod.ActiveDirectoryInteractive,
                //SC.SqlAuthenticationMethod.ActiveDirectoryIntegrated, // Alternatives.
                //SC.SqlAuthenticationMethod.ActiveDirectoryPassword,
                provider);

            Program.Connection();
        }

        public static void Connection()
        {
            SC.SqlConnectionStringBuilder builder = new SC.SqlConnectionStringBuilder();

            // Program._ static values that you set earlier.
            builder["Data Source"] = Program.Az_SQLDB_svrName;
            builder.UserID = Program.AzureAD_UserID;
            builder["Initial Catalog"] = Program.Initial_DatabaseName;

            // This "Password" is not used with .ActiveDirectoryInteractive.
            //builder["Password"] = "<YOUR PASSWORD HERE>";

            builder["Connect Timeout"] = 15;
            builder["TrustServerCertificate"] = true;
            builder.Pooling = false;

            // Assigned enum value must match the enum given to .SetProvider().
            builder.Authentication = SC.SqlAuthenticationMethod.ActiveDirectoryInteractive;
            SC.SqlConnection sqlConnection = new SC.SqlConnection(builder.ConnectionString);

            SC.SqlCommand cmd = new SC.SqlCommand(
                "SELECT '***** MY QUERY RAN SUCCESSFULLY!! *****';",
                sqlConnection);

            try
            {
                sqlConnection.Open();
                cmd.ExecuteNonQuery();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

```

    {
        sqlConnection.Open();
        if (sqlConnection.State == DA.ConnectionState.Open)
        {
            var rdr = cmd.ExecuteReader();
            var msg = new TX.StringBuilder();
            while (rdr.Read())
            {
                msg.AppendLine(rdr.GetString(0));
            }
            Console.WriteLine(msg.ToString());
            Console.WriteLine(":Success");
        }
        else
        {
            Console.WriteLine(":Failed");
        }
        sqlConnection.Close();
    }
    catch (Exception ex)
    {
        Console.ForegroundColor = ConsoleColor.Red;
        Console.WriteLine("Connection failed with the following exception...");
        Console.WriteLine(ex.ToString());
        Console.ResetColor();
    }
}
}

} // EOClass Program.

/// <summary>
/// SqlAuthenticationProvider - Is a public class that defines 3 different Azure AD
/// authentication methods. The methods are supported in the new .NET 4.7.2.
/// .
/// 1. Interactive, 2. Integrated, 3. Password
/// .
/// All 3 authentication methods are based on the Azure
/// Active Directory Authentication Library (ADAL) managed library.
/// </summary>
public class ActiveDirectoryAuthProvider : SC.SqlAuthenticationProvider
{
    // Program._ more static values that you set!
    private readonly string _clientId = Program.ClientApplicationID;
    private readonly Uri _redirectUri = Program.RedirectUri;

    public override async TT.Task<SC.SqlAuthenticationToken>
        AcquireTokenAsync(SC.SqlAuthenticationParameters parameters)
    {
        AD.AuthenticationContext authContext =
            new AD.AuthenticationContext(parameters.Authority);
        authContext.CorrelationId = parameters.ConnectionId;
        AD.AuthenticationResult result;

        switch (parameters.AuthenticationMethod)
        {
            case SC.SqlAuthenticationMethod.ActiveDirectoryInteractive:
                Console.WriteLine("In method 'AcquireTokenAsync', case_0 ==
'.ActiveDirectoryInteractive'.");
                result = await authContext.AcquireTokenAsync(
                    parameters.Resource, // "https://database.windows.net/"
                    _clientId,
                    _redirectUri,
                    new AD.PlatformParameters(AD.PromptBehavior.Auto),
                    new AD.UserIdentifier(
                        parameters.UserId,
                        AD.UserIdentifierType.RequiredDisplayableId));
                break;

            case SC.SqlAuthenticationMethod.ActiveDirectoryIntegrated:

```

```

        Console.WriteLine("In method 'AcquireTokenAsync', case_1 ==
'.ActiveDirectoryIntegrated'.");

        result = await authContext.AcquireTokenAsync(
            parameters.Resource,
            _clientId,
            new AD.UserCredential());
        break;

    case SC.SqlAuthenticationMethod.ActiveDirectoryPassword:
        Console.WriteLine("In method 'AcquireTokenAsync', case_2 == '.ActiveDirectoryPassword'.");

        result = await authContext.AcquireTokenAsync(
            parameters.Resource,
            _clientId,
            new AD.UserPasswordCredential(
                parameters.UserId,
                parameters.Password));
        break;

    default: throw new InvalidOperationException();
}
return new SC.SqlAuthenticationToken(result.AccessToken, result.ExpiresOn);
}

public override bool IsSupported(SC.SqlAuthenticationMethod authenticationMethod)
{
    return authenticationMethod == SC.SqlAuthenticationMethod.ActiveDirectoryIntegrated
        || authenticationMethod == SC.SqlAuthenticationMethod.ActiveDirectoryInteractive
        || authenticationMethod == SC.SqlAuthenticationMethod.ActiveDirectoryPassword;
}

} // EOClass ActiveDirectoryAuthProvider.
} // EONamespace. End of entire program source code.

```

This is an example of the C# test output.

```

[C:\Test\VSProj\ADInteractive5\ADInteractive5\bin\Debug\]
>> ADInteractive5.exe
In method 'AcquireTokenAsync', case_0 == '.ActiveDirectoryInteractive'.
***** MY QUERY RAN SUCCESSFULLY!! *****

:Success

[C:\Test\VSProj\ADInteractive5\ADInteractive5\bin\Debug\]
>>

```

Next steps

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

- [Get-AzSqlServerActiveDirectoryAdministrator](#)

Quickstart: Use Golang to query an Azure SQL database

11/7/2019 • 5 minutes to read • [Edit Online](#)

In this quickstart, you'll use the [Golang](#) programming language to connect to an Azure SQL database. You'll then run Transact-SQL statements to query and modify data. [Golang](#) is an open-source programming language that makes it easy to build simple, reliable, and efficient software.

Prerequisites

To complete this tutorial, you need:

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

- Golang and related software for your operating system installed:
 - **MacOS:** Install Homebrew and Golang. See [Step 1.2](#).
 - **Ubuntu:** Install Golang. See [Step 1.2](#).
 - **Windows:** Install Golang. See [Step 1.2](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified

server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create Golang project and dependencies

1. From the terminal, create a new project folder called **SqlServerSample**.

```
mkdir SqlServerSample
```

2. Navigate to **SqlServerSample** and install the SQL Server driver for Go.

```
cd SqlServerSample
go get github.com/denisenkom/go-mssql
go install github.com/denisenkom/go-mssql
```

Create sample data

1. In a text editor, create a file called **CreateTestData.sql** in the **SqlServerSample** folder. In the file, paste this T-SQL code, which creates a schema, table, and inserts a few rows.

```
CREATE SCHEMA TestSchema;
GO

CREATE TABLE TestSchema.Employees (
    Id      INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
    Name    NVARCHAR(50),
    Location NVARCHAR(50)
);
GO

INSERT INTO TestSchema.Employees (Name, Location) VALUES
    (N'Jared', N'Australia'),
    (N'Nikita', N'India'),
    (N'Tom', N'Germany');
GO

SELECT * FROM TestSchema.Employees;
GO
```

2. Use `sqlcmd` to connect to the database and run your newly created SQL script. Replace the appropriate values for your server, database, username, and password.

```
sqlcmd -S <your_server>.database.windows.net -U <your_username> -P <your_password> -d <your_database> -i
./CreateTestData.sql
```

Insert code to query SQL database

1. Create a file named **sample.go** in the **SqlServerSample** folder.

2. In the file, paste this code. Add the values for your server, database, username, and password. This example uses the Golang [context methods](#) to make sure there's an active database server connection.

```
package main

import (
    _ "github.com/denisenkom/go-mssqldb"
    "database/sql"
    "context"
    "log"
    "fmt"
    "errors"
)

var db *sql.DB

var server = "<your_server.database.windows.net>"
var port = 1433
var user = "<your_username>"
var password = "<your_password>"
var database = "<your_database>"

func main() {
    // Build connection string
    connString := fmt.Sprintf("server=%s;user id=%s;password=%s;port=%d;database=%s;",
        server, user, password, port, database)

    var err error

    // Create connection pool
    db, err = sql.Open("sqlserver", connString)
    if err != nil {
        log.Fatal("Error creating connection pool: ", err.Error())
    }
    ctx := context.Background()
    err = db.PingContext(ctx)
    if err != nil {
        log.Fatal(err.Error())
    }
    fmt.Printf("Connected!\n")

    // Create employee
    createID, err := CreateEmployee("Jake", "United States")
    if err != nil {
        log.Fatal("Error creating Employee: ", err.Error())
    }
    fmt.Printf("Inserted ID: %d successfully.\n", createID)

    // Read employees
    count, err := ReadEmployees()
    if err != nil {
        log.Fatal("Error reading Employees: ", err.Error())
    }
    fmt.Printf("Read %d row(s) successfully.\n", count)

    // Update from database
    updatedRows, err := UpdateEmployee("Jake", "Poland")
    if err != nil {
        log.Fatal("Error updating Employee: ", err.Error())
    }
    fmt.Printf("Updated %d row(s) successfully.\n", updatedRows)

    // Delete from database
    deletedRows, err := DeleteEmployee("Jake")
    if err != nil {
        log.Fatal("Error deleting Employee: ", err.Error())
    }
}
```

```

        fmt.Println("Deleted %d row(s) successfully.\n", deleteRows)
    }

// CreateEmployee inserts an employee record
func CreateEmployee(name string, location string) (int64, error) {
    ctx := context.Background()
    var err error

    if db == nil {
        err = errors.New("CreateEmployee: db is null")
        return -1, err
    }

    // Check if database is alive.
    err = db.PingContext(ctx)
    if err != nil {
        return -1, err
    }

    tsql := "INSERT INTO TestSchema.Employees (Name, Location) VALUES (@Name, @Location); select convert(bigint, SCOPE_IDENTITY());"

    stmt, err := db.Prepare(tsql)
    if err != nil {
        return -1, err
    }
    defer stmt.Close()

    row := stmt.QueryRowContext(
        ctx,
        sql.Named("Name", name),
        sql.Named("Location", location))
    var newID int64
    err = row.Scan(&newID)
    if err != nil {
        return -1, err
    }

    return newID, nil
}

// ReadEmployees reads all employee records
func ReadEmployees() (int, error) {
    ctx := context.Background()

    // Check if database is alive.
    err := db.PingContext(ctx)
    if err != nil {
        return -1, err
    }

    tsql := fmt.Sprintf("SELECT Id, Name, Location FROM TestSchema.Employees;")

    // Execute query
    rows, err := db.QueryContext(ctx, tsql)
    if err != nil {
        return -1, err
    }

    defer rows.Close()

    var count int

    // Iterate through the result set.
    for rows.Next() {
        var name, location string
        var id int

        // Get values from row.

```

```

        err := rows.Scan(&id, &name, &location)
        if err != nil {
            return -1, err
        }

        fmt.Printf("ID: %d, Name: %s, Location: %s\n", id, name, location)
        count++
    }

    return count, nil
}

// UpdateEmployee updates an employee's information
func UpdateEmployee(name string, location string) (int64, error) {
    ctx := context.Background()

    // Check if database is alive.
    err := db.PingContext(ctx)
    if err != nil {
        return -1, err
    }

    tsq1 := fmt.Sprintf("UPDATE TestSchema.Employees SET Location = @Location WHERE Name = @Name")

    // Execute non-query with named parameters
    result, err := db.ExecContext(
        ctx,
        tsq1,
        sql.Named("Location", location),
        sql.Named("Name", name))
    if err != nil {
        return -1, err
    }

    return result.RowsAffected()
}

// DeleteEmployee deletes an employee from the database
func DeleteEmployee(name string) (int64, error) {
    ctx := context.Background()

    // Check if database is alive.
    err := db.PingContext(ctx)
    if err != nil {
        return -1, err
    }

    tsq1 := fmt.Sprintf("DELETE FROM TestSchema.Employees WHERE Name = @Name;")

    // Execute non-query with named parameters
    result, err := db.ExecContext(ctx, tsq1, sql.Named("Name", name))
    if err != nil {
        return -1, err
    }

    return result.RowsAffected()
}

```

Run the code

- At the command prompt, run the following command.

```
go run sample.go
```

- Verify the output.

```
Connected!
Inserted ID: 4 successfully.
ID: 1, Name: Jared, Location: Australia
ID: 2, Name: Nikita, Location: India
ID: 3, Name: Tom, Location: Germany
ID: 4, Name: Jake, Location: United States
Read 4 row(s) successfully.
Updated 1 row(s) successfully.
Deleted 1 row(s) successfully.
```

Next steps

- [Design your first Azure SQL database](#)
- [Golang driver for Microsoft SQL Server](#)
- [Report issues or ask questions](#)

Quickstart: Use Java to query an Azure SQL database

1/26/2020 • 3 minutes to read • [Edit Online](#)

In this quickstart, you use Java to connect to an Azure SQL database and use T-SQL statements to query data.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An [Azure SQL database](#)
- [Java](#)-related software
 - macOS
 - Ubuntu
 - Windows

Install Homebrew and Java, then install Maven using steps **1.2** and **1.3** in [Create Java apps using SQL Server on macOS](#).

IMPORTANT

The scripts in this article are written to use the **Adventure Works** database.

NOTE

You can optionally choose to use an Azure SQL managed instance.

To create and configure, use the [Azure Portal](#), [PowerShell](#), or [CLI](#), then setup [on-site](#) or [VM](#) connectivity.

To load data, see [restore with BACPAC](#) with the [Adventure Works](#) file, or see [restore the Wide World Importers database](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL databases** or open the **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create the project

1. From the command prompt, create a new Maven project called *sqltest*.

```
mvn archetype:generate "-DgroupId=com.sql dbsamples" "-DartifactId=sqltest" "-DarchetypeArtifactId=maven-archetype-quickstart" "-Dversion=1.0.0" --batch-mode
```

2. Change the folder to *sqltest* and open *pom.xml* with your favorite text editor. Add the **Microsoft JDBC Driver for SQL Server** to your project's dependencies using the following code.

```
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <version>7.0.0.jre8</version>
</dependency>
```

3. Also in *pom.xml*, add the following properties to your project. If you don't have a properties section, you can add it after the dependencies.

```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

4. Save and close *pom.xml*.

Add code to query database

1. You should already have a file called *App.java* in your Maven project located at:

..\\sqltest\\src\\main\\java\\com\\sql dbsamples\\App.java

2. Open the file and replace its contents with the following code. Then add the appropriate values for your server, database, user, and password.

```

package com.sql dbsamples;

import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.DriverManager;

public class App {

    public static void main(String[] args) {

        // Connect to database
        String hostName = "your_server.database.windows.net"; // update me
        String dbName = "your_database"; // update me
        String user = "your_username"; // update me
        String password = "your_password"; // update me
        String url =
String.format("jdbc:sqlserver://%s:1433;database=%s;user=%s;password=%s;encrypt=true;" +
            + "hostNameInCertificate=*.database.windows.net;loginTimeout=30;", hostName, dbName, user,
password);
        Connection connection = null;

        try {
            connection = DriverManager.getConnection(url);
            String schema = connection.getSchema();
            System.out.println("Successful connection - Schema: " + schema);

            System.out.println("Query data example:");
            System.out.println("=====");

            // Create and execute a SELECT SQL statement.
            String selectSql = "SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName " +
                + "FROM [SalesLT].[ProductCategory] pc " +
                + "JOIN [SalesLT].[Product] p ON pc.productcategoryid = p.productcategoryid";

            try (Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(selectSql)) {

                // Print results from select statement
                System.out.println("Top 20 categories:");
                while (resultSet.next())
                {
                    System.out.println(resultSet.getString(1) + " " +
                        + resultSet.getString(2));
                }
                connection.close();
            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

NOTE

The code example uses the **AdventureWorksLT** sample database for Azure SQL.

Run the code

1. At the command prompt, run the app.

```
mvn package -DskipTests  
mvn -q exec:java "-Dexec.mainClass=com.sql dbsamples.App"
```

2. Verify the top 20 rows are returned and close the app window.

Next steps

- [Design your first Azure SQL database](#)
- [Microsoft JDBC Driver for SQL Server](#)
- [Report issues/ask questions](#)

Quickstart: Use Node.js to query an Azure SQL database

1/26/2020 • 2 minutes to read • [Edit Online](#)

In this quickstart, you use Node.js to connect to an Azure SQL database and use T-SQL statements to query data.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An [Azure SQL database](#)
- [Nodejs](#)-related software
 - [macOS](#)
 - [Ubuntu](#)
 - [Windows](#)

Install Homebrew and Node.js, then install the ODBC driver and SQLCMD using steps **1.2** and **1.3** in [Create Nodejs apps using SQL Server on macOS](#).

IMPORTANT

The scripts in this article are written to use the **Adventure Works** database.

NOTE

You can optionally choose to use an Azure SQL managed instance.

To create and configure, use the [Azure Portal](#), [PowerShell](#), or [CLI](#), then setup [on-site](#) or [VM](#) connectivity.

To load data, see [restore with BACPAC](#) with the [Adventure Works](#) file, or see [restore the Wide World Importers database](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Go to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create the project

Open a command prompt and create a folder named *sqltest*. Open the folder you created and run the following command:

```
npm init -y  
npm install tedious
```

Add code to query database

1. In your favorite text editor, create a new file, *sqltest.js*.
2. Replace its contents with the following code. Then add the appropriate values for your server, database, user, and password.

```

const { Connection, Request } = require("tedious");

// Create connection to database
const config = {
  authentication: {
    options: {
      userName: "username", // update me
      password: "password" // update me
    },
    type: "default"
  },
  server: "your_server.database.windows.net", // update me
  options: {
    database: "your_database", //update me
    encrypt: true
  }
};

const connection = new Connection(config);

// Attempt to connect and execute queries if connection goes through
connection.on("connect", err => {
  if (err) {
    console.error(err.message);
  } else {
    queryDatabase();
  }
});

function queryDatabase() {
  console.log("Reading rows from the Table...");

  // Read all rows from table
  const request = new Request(
    `SELECT TOP 20 pc.Name as CategoryName,
      p.name as ProductName
    FROM [SalesLT].[ProductCategory] pc
    JOIN [SalesLT].[Product] p ON pc.productcategoryid = p.productcategoryid`,
    (err, rowCount) => {
      if (err) {
        console.error(err.message);
      } else {
        console.log(` ${rowCount} row(s) returned`);
      }
    }
  );

  request.on("row", columns => {
    columns.forEach(column => {
      console.log("%s\t%s", column.metadata.colName, column.value);
    });
  });

  connection.execSql(request);
}

```

NOTE

The code example uses the **AdventureWorksLT** sample database for Azure SQL.

Run the code

1. At the command prompt, run the program.

```
node sqltest.js
```

2. Verify the top 20 rows are returned and close the application window.

Next steps

- [Microsoft Node.js Driver for SQL Server](#)
- Connect and query on Windows/Linux/macOS with [.NET core](#), [Visual Studio Code](#), or [SSMS](#) (Windows only)
- [Get started with .NET Core on Windows/Linux/macOS using the command line](#)
- Design your first Azure SQL database using [.NET](#) or [SSMS](#)

Quickstart: Use PHP to query an Azure SQL database

11/7/2019 • 2 minutes to read • [Edit Online](#)

This article demonstrates how to use [PHP](#) to connect to an Azure SQL database. You can then use T-SQL statements to query data.

Prerequisites

To complete this sample, make sure you have the following prerequisites:

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

- PHP-related software installed for your operating system:
 - **MacOS**, install PHP, the ODBC driver, then install the PHP Driver for SQL Server. See [Step 1, 2, and 3](#).
 - **Linux**, install PHP, the ODBC driver, then install the PHP Driver for SQL Server. See [Step 1, 2, and 3](#).
 - **Windows**, install PHP for IIS Express and Chocolatey, then install the ODBC driver and SQLCMD. See [Step 1.2 and 1.3](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Add code to query database

1. In your favorite text editor, create a new file, `sqltest.php`.
2. Replace its contents with the following code. Then add the appropriate values for your server, database, user, and password.

```
<?php
$serverName = "your_server.database.windows.net"; // update me
$connectionOptions = array(
    "Database" => "your_database", // update me
    "Uid" => "your_username", // update me
    "PWD" => "your_password" // update me
);
//Establishes the connection
$conn = sqlsrv_connect($serverName, $connectionOptions);
$tsql= "SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
        FROM [SalesLT].[ProductCategory] pc
        JOIN [SalesLT].[Product] p
        ON pc.productcategoryid = p.productcategoryid";
$getResults= sqlsrv_query($conn, $tsql);
echo ("Reading data from table" . PHP_EOL);
if ($getResults == FALSE)
    echo (sqlsrv_errors());
while ($row = sqlsrv_fetch_array($getResults, SQLSRV_FETCH_ASSOC)) {
    echo ($row['CategoryName'] . " " . $row['ProductName'] . PHP_EOL);
}
sqlsrv_free_stmt($getResults);
?>
```

Run the code

1. At the command prompt, run the app.

```
php sqltest.php
```

2. Verify the top 20 rows are returned and close the app window.

Next steps

- [Design your first Azure SQL database](#)
- [Microsoft PHP Drivers for SQL Server](#)
- [Report issues or ask questions](#)

- Retry logic example: Connect resiliently to SQL with PHP

Quickstart: Use Python to query an Azure SQL database

1/26/2020 • 2 minutes to read • [Edit Online](#)

In this quickstart, you use Python to connect to an Azure SQL database and use T-SQL statements to query data.

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An [Azure SQL database](#)
- [Python](#) 3 and related software
 - [macOS](#)
 - [Ubuntu](#)
 - [Windows](#)

To install Homebrew and Python, the ODBC driver and SQLCMD, and the Python driver for SQL Server, use steps **1.2**, **1.3**, and **2.1** in [create Python apps using SQL Server on macOS](#).

For further information, see [Microsoft ODBC Driver on macOS](#).

IMPORTANT

The scripts in this article are written to use the **Adventure Works** database.

NOTE

You can optionally choose to use an Azure SQL managed instance.

To create and configure, use the [Azure Portal](#), [PowerShell](#), or [CLI](#), then setup [on-site](#) or [VM](#) connectivity.

To load data, see [restore with BACPAC](#) with the [Adventure Works](#) file, or see [restore the Wide World Importers database](#).

To further explore Python and the Azure SQL database, see [Azure SQL database libraries for Python](#), the [pyodbc repository](#), and a [pyodbc sample](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Go to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create code to query your SQL database

1. In a text editor, create a new file named `sqltest.py`.
2. Add the following code. Substitute your own values for <server>, <database>, <username>, and <password>.

IMPORTANT

The code in this example uses the sample AdventureWorksLT data, which you can choose as source when creating your database. If your database has different data, use tables from your own database in the SELECT query.

```
import pyodbc
server = '<server>.database.windows.net'
database = '<database>'
username = '<username>'
password = '<password>'
driver= '{ODBC Driver 17 for SQL Server}'
cnxn =
pyodbc.connect('DRIVER=' + driver +';SERVER=' + server +';PORT=1433;DATABASE=' + database +';UID=' + username +';PWD=' + password)
cursor = cnxn.cursor()
cursor.execute("SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName FROM [SalesLT].[ProductCategory] pc JOIN [SalesLT].[Product] p ON pc.productcategoryid = p.productcategoryid")
row = cursor.fetchone()
while row:
    print (str(row[0]) + " " + str(row[1]))
    row = cursor.fetchone()
```

Run the code

1. At a command prompt, run the following command:

```
python sqltest.py
```

2. Verify that the top 20 Category/Product rows are returned, then close the command window.

Next steps

- [Design your first Azure SQL database](#)
- [Microsoft Python drivers for SQL Server](#)
- [Python developer center](#)

Quickstart: Use Ruby to query an Azure SQL database

11/7/2019 • 2 minutes to read • [Edit Online](#)

This quickstart demonstrates how to use [Ruby](#) to connect to an Azure SQL database and query data with Transact-SQL statements.

Prerequisites

To complete this quickstart, you need the following prerequisites:

- An Azure SQL database. You can use one of these quickstarts to create and then configure a database in Azure SQL Database:

	SINGLE DATABASE	MANAGED INSTANCE
Create	Portal	Portal
	CLI	CLI
	PowerShell	PowerShell
Configure	Server-level IP firewall rule	Connectivity from a VM
		Connectivity from on-site
Load data	Adventure Works loaded per quickstart	Restore Wide World Importers
		Restore or import Adventure Works from BACPAC file from GitHub

IMPORTANT

The scripts in this article are written to use the Adventure Works database. With a managed instance, you must either import the Adventure Works database into an instance database or modify the scripts in this article to use the Wide World Importers database.

- Ruby and related software for your operating system:
 - **MacOS:** Install Homebrew, rbenv and ruby-build, Ruby, FreeTDS, and TinyTDS. See Steps 1.2, 1.3, 1.4, 1.5, and 2.1 in [Create Ruby apps using SQL Server on macOS](#).
 - **Ubuntu:** Install prerequisites for Ruby, rbenv and ruby-build, Ruby, FreeTDS, and TinyTDS. See Steps 1.2, 1.3, 1.4, 1.5, and 2.1 in [Create Ruby apps using SQL Server on Ubuntu](#).
 - **Windows:** Install Ruby, Ruby Devkit, and TinyTDS. See [Configure development environment for Ruby development](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create code to query your SQL database

1. In a text or code editor, create a new file named `sqltest.rb`.
2. Add the following code. Substitute the values from your Azure SQL database for `<server>`, `<database>`, `<username>`, and `<password>`.

IMPORTANT

The code in this example uses the sample AdventureWorksLT data, which you can choose as source when creating your database. If your database has different data, use tables from your own database in the SELECT query.

```
require 'tiny_tds'
server = '<server>.database.windows.net'
database = '<database>'
username = '<username>'
password = '<password>'
client = TinyTds::Client.new username: username, password: password,
                           host: server, port: 1433, database: database, azure: true

puts "Reading data from table"
tsql = "SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
        FROM [SalesLT].[ProductCategory] pc
        JOIN [SalesLT].[Product] p
        ON pc.productcategoryid = p.productcategoryid"
result = client.execute(tsql)
result.each do |row|
  puts row
end
```

Run the code

1. At a command prompt, run the following command:

```
ruby sqltest.rb
```

2. Verify that the top 20 Category/Product rows from your database are returned.

Next steps

- [Design your first Azure SQL database](#).
- [GitHub repository for TinyTDS](#).
- [Report issues or ask questions about TinyTDS](#).

- [Ruby driver for SQL Server.](#)

Quickstart: Use R with Machine Learning Services to query an Azure SQL database (preview)

1/26/2020 • 2 minutes to read • [Edit Online](#)

In this quickstart, you use R with Machine Learning Services to connect to an Azure SQL database and use T-SQL statements to query data.

IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An [Azure SQL database](#)
- [Machine Learning Services](#) with R enabled. [Sign up for the preview](#).
- [SQL Server Management Studio \(SSMS\)](#)

IMPORTANT

The scripts in this article are written to use the **Adventure Works** database.

NOTE

During the public preview, Microsoft will onboard you and enable machine learning for your existing or new database, however the managed instance deployment option is currently not supported.

Machine Learning Services with R is a feature of Azure SQL database used for executing in-database R scripts. For more information, see the [R Project](#).

Get SQL server connection information

Get the connection information you need to connect to the Azure SQL database. You'll need the fully qualified server name or host name, database name, and login information for the upcoming procedures.

1. Sign in to the [Azure portal](#).
2. Navigate to the **SQL databases** or **SQL managed instances** page.
3. On the **Overview** page, review the fully qualified server name next to **Server name** for a single database or the fully qualified server name next to **Host** for a managed instance. To copy the server name or host name, hover over it and select the **Copy** icon.

Create code to query your SQL database

1. Open **SQL Server Management Studio** and connect to your SQL database.

If you need help connecting, see [Quickstart: Use SQL Server Management Studio to connect and query an Azure SQL database](#).

2. Pass the complete R script to the `sp_execute_external_script` stored procedure.

The script is passed through the `@script` argument. Everything inside the `@script` argument must be valid R code.

IMPORTANT

The code in this example uses the sample AdventureWorksLT data, which you can choose as source when creating your database. If your database has different data, use tables from your own database in the SELECT query.

```
EXECUTE sp_execute_external_script
@language = N'R'
, @script = N'OutputDataSet <- InputDataSet;'
, @input_data_1 = N'SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName FROM [SalesLT].[ProductCategory] pc JOIN [SalesLT].[Product] p ON pc.productcategoryid = p.productcategoryid'
```

NOTE

If you get any errors, it might be because the public preview of Machine Learning Services (with R) is not enabled for your SQL database. See [Prerequisites](#) above.

Run the code

1. Execute the `sp_execute_external_script` stored procedure.
2. Verify that the top 20 Category/Product rows are returned in the **Messages** window.

Next steps

- [Design your first Azure SQL database](#)
- [Azure SQL Database Machine Learning Services \(with R\)](#)
- [Create and run simple R scripts in Azure SQL Database Machine Learning Services \(preview\)](#)
- [Write advanced R functions in Azure SQL Database using Machine Learning Services \(preview\)](#)

Overview of business continuity with Azure SQL Database

11/7/2019 • 9 minutes to read • [Edit Online](#)

Business continuity in Azure SQL Database refers to the mechanisms, policies, and procedures that enable your business to continue operating in the face of disruption, particularly to its computing infrastructure. In the most of the cases, Azure SQL Database will handle the disruptive events that might happen in the cloud environment and keep your applications and business processes running. However, there are some disruptive events that cannot be handled by SQL Database automatically such as:

- User accidentally deleted or updated a row in a table.
- Malicious attacker succeeded to delete data or drop a database.
- Earthquake caused a power outage and temporary disabled data-center.

This overview describes the capabilities that Azure SQL Database provides for business continuity and disaster recovery. Learn about options, recommendations, and tutorials for recovering from disruptive events that could cause data loss or cause your database and application to become unavailable. Learn what to do when a user or application error affects data integrity, an Azure region has an outage, or your application requires maintenance.

SQL Database features that you can use to provide business continuity

From a database perspective, there are four major potential disruption scenarios:

- Local hardware or software failures affecting the database node such as a disk-drive failure.
- Data corruption or deletion typically caused by an application bug or human error. Such failures are application-specific and typically cannot be detected by the database service.
- Datacenter outage, possibly caused by a natural disaster. This scenario requires some level of geo-redundancy with application failover to an alternate datacenter.
- Upgrade or maintenance errors, unanticipated issues that occur during planned infrastructure maintenance or upgrades may require rapid rollback to a prior database state.

To mitigate the local hardware and software failures, SQL Database includes a [high availability architecture](#), which guarantees automatic recovery from these failures with up to 99.995% availability SLA.

To protect your business from data loss, SQL Database automatically creates full database backups weekly, differential database backups every 12 hours, and transaction log backups every 5 - 10 minutes . The backups are stored in RA-GRS storage for at least 7 days for all service tiers. All service tiers except Basic support configurable backup retention period for point-in-time restore, up to 35 days.

SQL Database also provides several business continuity features, that you can use to mitigate various unplanned scenarios.

- [Temporal tables](#) enable you to restore row versions from any point in time.
- [Built-in automated backups](#) and [Point in Time Restore](#) enables you to restore complete database to some point in time within the configured retention period up to 35 days.
- You can [restore a deleted database](#) to the point at which it was deleted if the **SQL Database server has not been deleted**.

- [Long-term backup retention](#) enables you to keep the backups up to 10 years.
- [Active geo-replication](#) enables you to create readable replicas and manually failover to any replica in case of a data center outage or application upgrade.
- [Auto-failover group](#) allows the application to automatically recover in case of a data center outage.

Recover a database within the same Azure region

You can use automatic database backups to restore a database to a point in time in the past. This way you can recover from data corruptions caused by human errors. The point-in-time restore allows you to create a new database in the same server that represents the state of data prior to the corrupting event. For most databases the restore operations takes less than 12 hours. It may take longer to recover a very large or very active database. For more information about recovery time, see [database recovery time](#).

If the maximum supported backup retention period for point-in-time restore (PITR) is not sufficient for your application, you can extend it by configuring a long-term retention (LTR) policy for the database(s). For more information, see [Long-term backup retention](#).

Compare geo-replication with failover groups

[Auto-failover groups](#) simplify the deployment and usage of [geo-replication](#) and add the additional capabilities as described in the following table:

	GEO-REPLICATION	FAILOVER GROUPS
Automatic failover	No	Yes
Fail over multiple databases simultaneously	No	Yes
Update connection string after failover	Yes	No
Managed instance supported	No	Yes
Can be in same region as primary	Yes	No
Multiple replicas	Yes	No
Supports read-scale	Yes	Yes

Recover a database to the existing server

Although rare, an Azure data center can have an outage. When an outage occurs, it causes a business disruption that might only last a few minutes or might last for hours.

- One option is to wait for your database to come back online when the data center outage is over. This works for applications that can afford to have the database offline. For example, a development project or free trial you don't need to work on constantly. When a data center has an outage, you do not know how long the outage might last, so this option only works if you don't need your database for a while.
- Another option is to restore a database on any server in any Azure region using [geo-redundant database backups](#) (geo-restore). Geo-restore uses a geo-redundant backup as its source and can be used to recover

a database even if the database or datacenter is inaccessible due to an outage.

- Finally, you can quickly recover from an outage if you have configured either geo-secondary using [active geo-replication](#) or an [auto-failover group](#) for your database or databases. Depending on your choice of these technologies, you can use either manual or automatic failover. While failover itself takes only a few seconds, the service will take at least 1 hour to activate it. This is necessary to ensure that the failover is justified by the scale of the outage. Also, the failover may result in small data loss due to the nature of asynchronous replication.

As you develop your business continuity plan, you need to understand the maximum acceptable time before the application fully recovers after the disruptive event. The time required for application to fully recover is known as Recovery time objective (RTO). You also need to understand the maximum period of recent data updates (time interval) the application can tolerate losing when recovering from an unplanned disruptive event. The potential data loss is known as Recovery point objective (RPO).

Different recovery methods offer different levels of RPO and RTO. You can choose a specific recovery method, or use a combination of methods to achieve full application recovery. The following table compares RPO and RTO of each recovery option. Auto-failover groups simplify the deployment and usage of geo-replication and adds the additional capabilities as described in the following table.

RECOVERY METHOD	RTO	RPO
Geo-restore from geo-replicated backups	12 h	1 h
Auto-failover groups	1 h	5 s
Manual database failover	30 s	5 s

NOTE

Manual database failover refers to failover of a single database to its geo-replicated secondary using the [unplanned mode](#). See the table earlier in this article for details of the auto-failover RTO and RPO.

Use auto-failover groups if your application meets any of these criteria:

- Is mission critical.
- Has a service level agreement (SLA) that does not allow for 12 hours or more of downtime.
- Downtime may result in financial liability.
- Has a high rate of data change and 1 hour of data loss is not acceptable.
- The additional cost of active geo-replication is lower than the potential financial liability and associated loss of business.

You may choose to use a combination of database backups and active geo-replication depending upon your application requirements. For a discussion of design considerations for stand-alone databases and for elastic pools using these business continuity features, see [Design an application for cloud disaster recovery](#) and [Elastic pool disaster recovery strategies](#).

The following sections provide an overview of the steps to recover using either database backups or active geo-replication. For detailed steps including planning requirements, post recovery steps, and information about how to simulate an outage to perform a disaster recovery drill, see [Recover a SQL Database from an](#)

[outage](#).

Prepare for an outage

Regardless of the business continuity feature you use, you must:

- Identify and prepare the target server, including server-level IP firewall rules, logins, and master database level permissions.
- Determine how to redirect clients and client applications to the new server
- Document other dependencies, such as auditing settings and alerts

If you do not prepare properly, bringing your applications online after a failover or a database recovery takes additional time and likely also require troubleshooting at a time of stress - a bad combination.

Fail over to a geo-replicated secondary database

If you are using active geo-replication or auto-failover groups as your recovery mechanism, you can configure an automatic failover policy or use [manual unplanned failover](#). Once initiated, the failover causes the secondary to become the new primary and ready to record new transactions and respond to queries - with minimal data loss for the data not yet replicated. For information on designing the failover process, see [Design an application for cloud disaster recovery](#).

NOTE

When the data center comes back online the old primaries automatically reconnect to the new primary and become secondary databases. If you need to relocate the primary back to the original region, you can initiate a planned failover manually (failback).

Perform a geo-restore

If you are using the automated backups with geo-redundant storage (enabled by default), you can recover the database using [geo-restore](#). Recovery usually takes place within 12 hours - with data loss of up to one hour determined by when the last log backup was taken and replicated. Until the recovery completes, the database is unable to record any transactions or respond to any queries. Note, geo-restore only restores the database to the last available point in time.

NOTE

If the data center comes back online before you switch your application over to the recovered database, you can cancel the recovery.

Perform post failover / recovery tasks

After recovery from either recovery mechanism, you must perform the following additional tasks before your users and applications are back up and running:

- Redirect clients and client applications to the new server and restored database
- Ensure appropriate server-level IP firewall rules are in place for users to connect or use [database-level firewalls](#) to enable appropriate rules.
- Ensure appropriate logins and master database level permissions are in place (or use [contained users](#))
- Configure auditing, as appropriate
- Configure alerts, as appropriate

NOTE

If you are using a failover group and connect to the databases using the read-write listener, the redirection after failover will happen automatically and transparently to the application.

Upgrade an application with minimal downtime

Sometimes an application must be taken offline because of planned maintenance such as an application upgrade. [Manage application upgrades](#) describes how to use active geo-replication to enable rolling upgrades of your cloud application to minimize downtime during upgrades and provide a recovery path if something goes wrong.

Next steps

For a discussion of application design considerations for stand-alone databases and for elastic pools, see [Design an application for cloud disaster recovery](#) and [Elastic pool disaster recovery strategies](#).

High-availability and Azure SQL Database

1/8/2020 • 9 minutes to read • [Edit Online](#)

The goal of the High Availability architecture in Azure SQL Database is to guarantee that your database is up and running minimum of 99.99% of time (For more information regarding specific SLA for different tiers, Please refer [SLA for Azure SQL Database](#)), without worrying about the impact of maintenance operations and outages. Azure automatically handles critical servicing tasks, such as patching, backups, Windows and SQL upgrades, as well as unplanned events such as underlying hardware, software, or network failures. When the underlying SQL instance is patched or fails over, the downtime is not noticeable if you [employ retry logic](#) in your app. Azure SQL Database can quickly recover even in the most critical circumstances ensuring that your data is always available.

The high availability solution is designed to ensure that committed data is never lost due to failures, that maintenance operations do not affect your workload, and that the database will not be a single point of failure in your software architecture. There are no maintenance windows or downtimes that should require you to stop the workload while the database is upgraded or maintained.

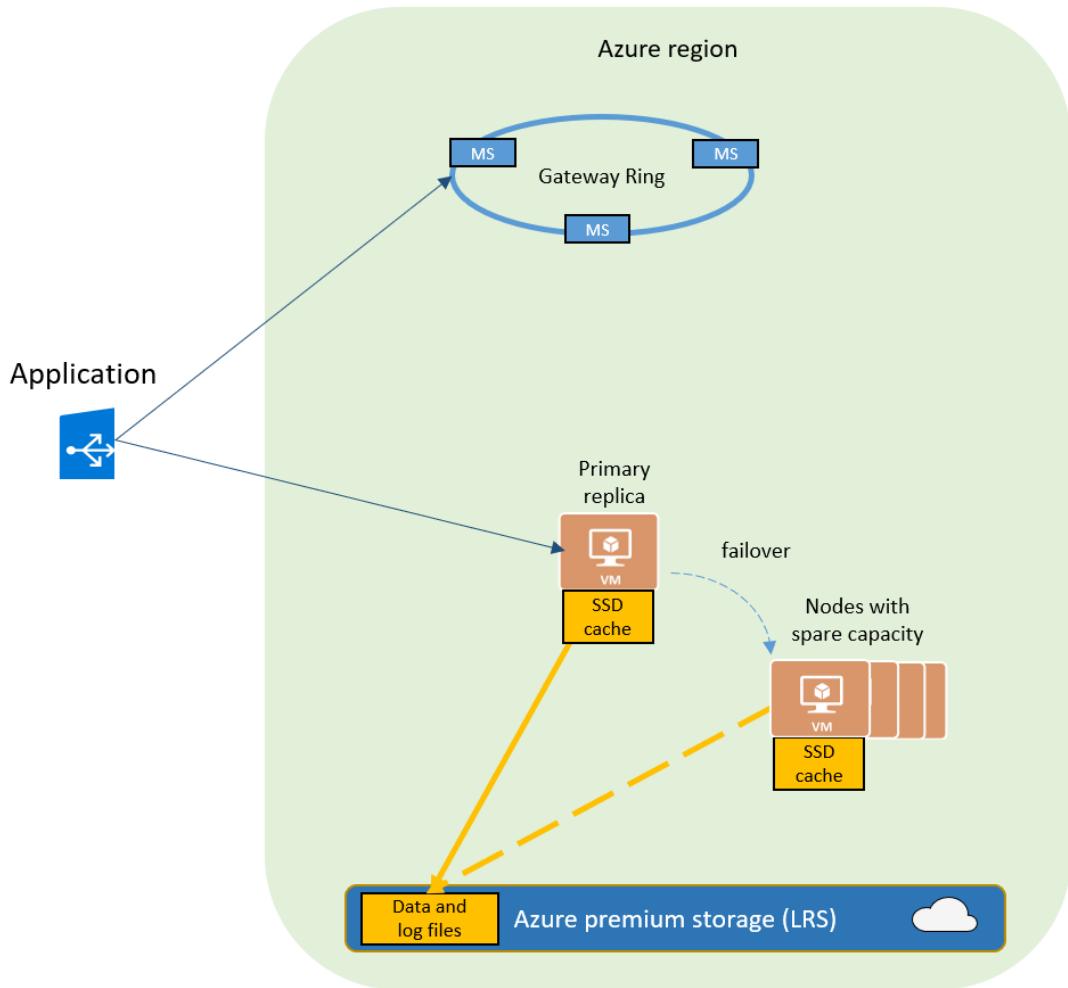
There are two high-availability architectural models that are used in Azure SQL Database:

- Standard availability model that is based on a separation of compute and storage. It relies on high availability and reliability of the remote storage tier. This architecture targets budget-oriented business applications that can tolerate some performance degradation during maintenance activities.
- Premium availability model that is based on a cluster of database engine processes. It relies on the fact that there is always a quorum of available database engine nodes. This architecture targets mission critical applications with high IO performance, high transaction rate and guarantees minimal performance impact to your workload during maintenance activities.

Azure SQL Database runs on the latest stable version of SQL Server Database Engine and Windows OS, and most users would not notice that upgrades are performed continuously.

Basic, Standard, and General Purpose service tier availability

These service tiers leverage the standard availability architecture. The following figure shows four different nodes with the separated compute and storage layers.



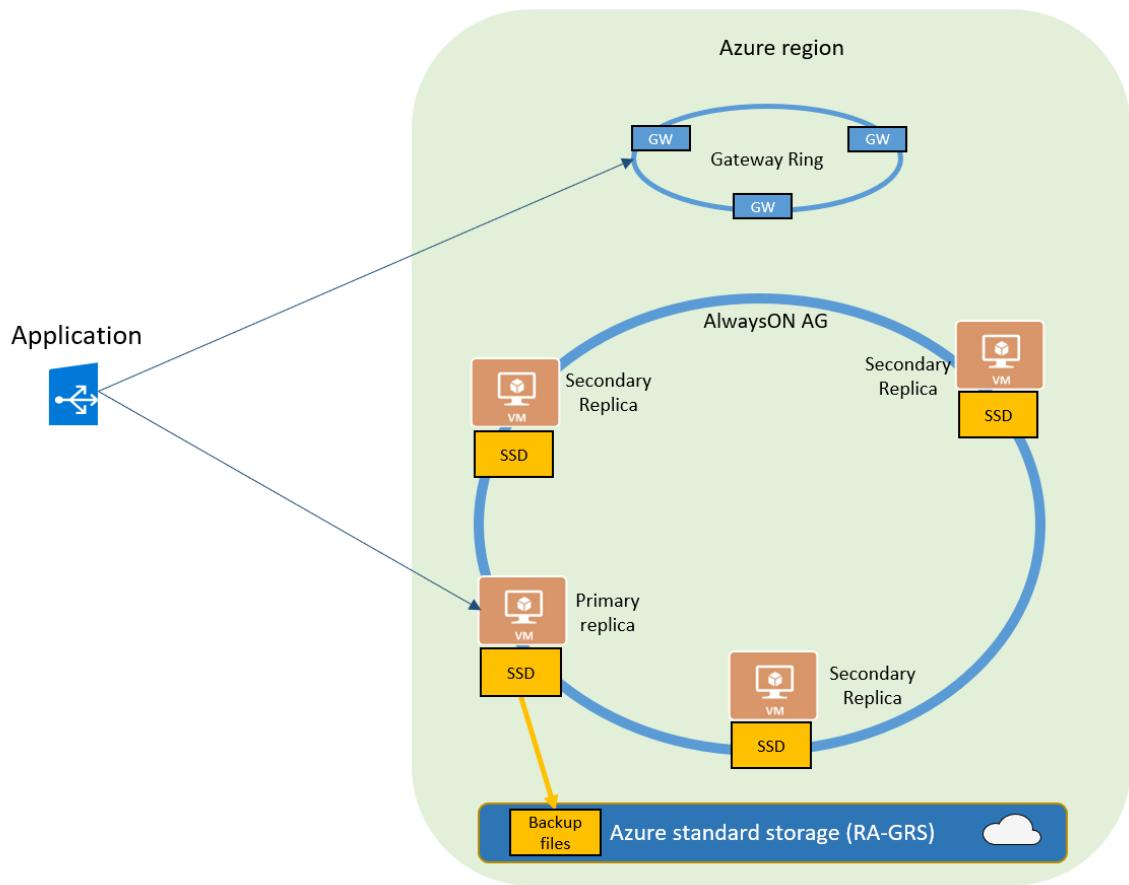
The standard availability model includes two layers:

- A stateless compute layer that runs the `sqlservr.exe` process and contains only transient and cached data, such as TempDB, model databases on the attached SSD, and plan cache, buffer pool, and columnstore pool in memory. This stateless node is operated by Azure Service Fabric that initializes `sqlservr.exe`, controls health of the node, and performs failover to another node if necessary.
- A stateful data layer with the database files (.mdf/.ldf) that are stored in Azure Blob storage. Azure blob storage has built-in data availability and redundancy feature. It guarantees that every record in the log file or page in the data file will be preserved even if SQL Server process crashes.

Whenever the database engine or the operating system is upgraded, or a failure is detected, Azure Service Fabric will move the stateless SQL Server process to another stateless compute node with sufficient free capacity. Data in Azure Blob storage is not affected by the move, and the data/log files are attached to the newly initialized SQL Server process. This process guarantees 99.99% availability, but a heavy workload may experience some performance degradation during the transition since the new SQL Server instance starts with cold cache.

Premium and Business Critical service tier availability

Premium and Business Critical service tiers leverage the Premium availability model, which integrates compute resources (SQL Server Database Engine process) and storage (locally attached SSD) on a single node. High availability is achieved by replicating both compute and storage to additional nodes creating a three to four-node cluster.

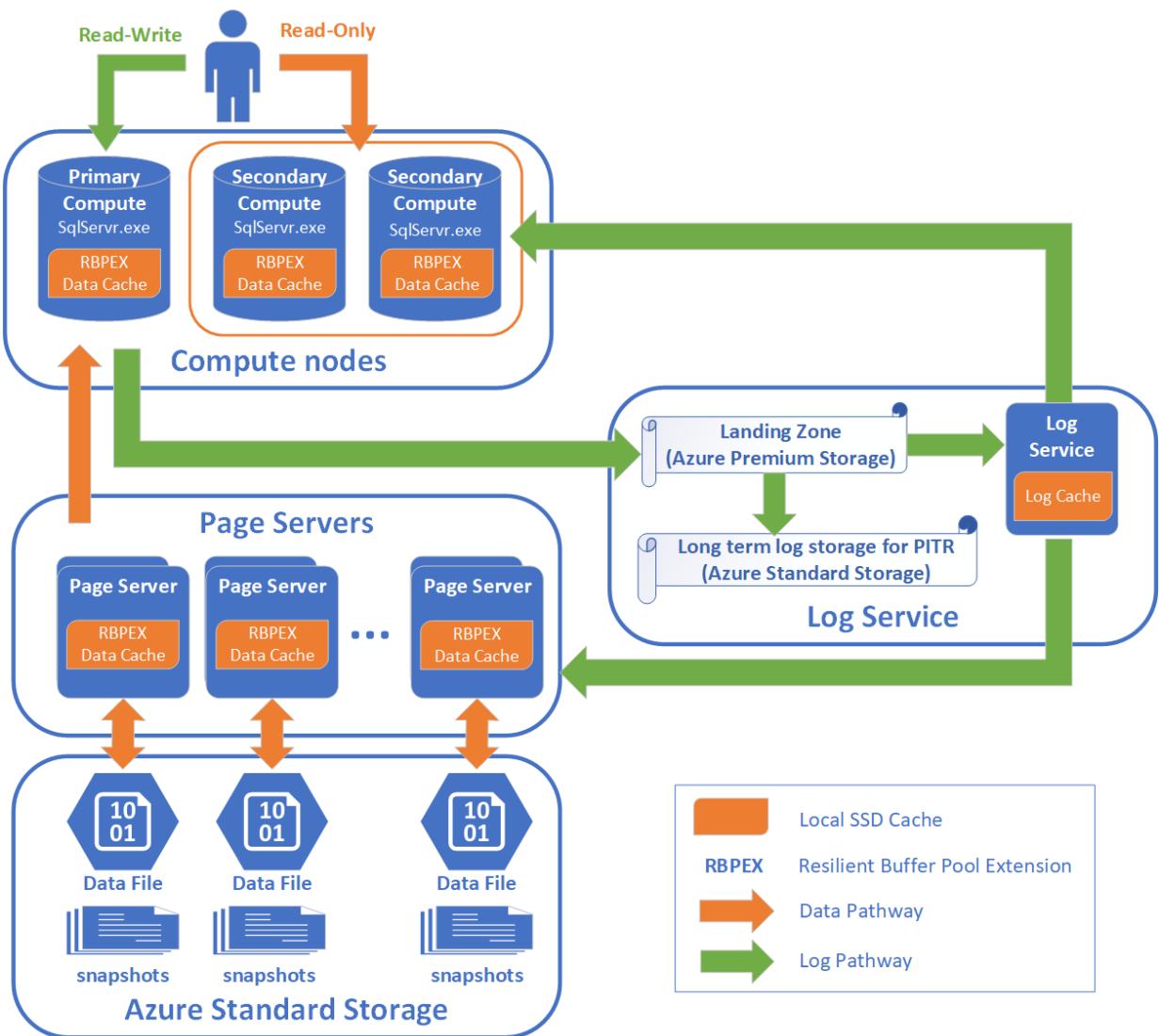


The underlying database files (.mdf/.ldf) are placed on the attached SSD storage to provide very low latency IO to your workload. High availability is implemented using a technology similar to SQL Server [Always On Availability Groups](#). The cluster includes a single primary replica (SQL Server process) that is accessible for read-write customer workloads, and up to three secondary replicas (compute and storage) containing copies of data. The primary node constantly pushes changes to the secondary nodes in order and ensures that the data is synchronized to at least one secondary replica before committing each transaction. This process guarantees that if the primary node crashes for any reason, there is always a fully synchronized node to fail over to. The failover is initiated by the Azure Service Fabric. Once the secondary replica becomes the new primary node, another secondary replica is created to ensure the cluster has enough nodes (quorum set). Once failover is complete, SQL connections are automatically redirected to the new primary node.

As an extra benefit, the premium availability model includes the ability to redirect read-only SQL connections to one of the secondary replicas. This feature is called [Read Scale-Out](#). It provides 100% additional compute capacity at no extra charge to off-load read-only operations, such as analytical workloads, from the primary replica.

Hyperscale service tier availability

The Hyperscale service tier architecture is described in [Distributed functions architecture](#).



The availability model in Hyperscale includes four layers:

- A stateless compute layer that runs the `sqlservr.exe` processes and contains only transient and cached data, such as non-covering RBPEX cache, TempDB, model database, etc. on the attached SSD, and plan cache, buffer pool, and columnstore pool in memory. This stateless layer includes the primary compute replica and optionally a number of secondary compute replicas that can serve as failover targets.
- A stateless storage layer formed by page servers. This layer is the distributed storage engine for the `sqlservr.exe` processes running on the compute replicas. Each page server contains only transient and cached data, such as covering RBPEX cache on the attached SSD, and data pages cached in memory. Each page server has a paired page server in an active-active configuration to provide load balancing, redundancy, and high availability.
- A stateful transaction log storage layer formed by the compute node running the Log service process, the transaction log landing zone, and transaction log long term storage. Landing zone and long term storage use Azure Storage, which provides availability and **redundancy** for transaction log, ensuring data durability for committed transactions.
- A stateful data storage layer with the database files (.mdf/.ndf) that are stored in Azure Storage and are updated by page servers. This layer uses data availability and **redundancy** features of Azure Storage. It guarantees that every page in a data file will be preserved even if processes in other layers of Hyperscale architecture crash, or if compute nodes fail.

Compute nodes in all Hyperscale layers run on Azure Service Fabric, which controls health of each node and performs failovers to available healthy nodes as necessary.

For more information on high availability in Hyperscale, see [Database High Availability in Hyperscale](#).

Zone redundant configuration

By default, the cluster of nodes for the premium availability model is created in the same datacenter. With the introduction of [Azure Availability Zones](#), SQL Database can place different replicas of the Business Critical database to different availability zones in the same region. To eliminate a single point of failure, the control ring is also duplicated across multiple zones as three gateway rings (GW). The routing to a specific gateway ring is controlled by [Azure Traffic Manager](#) (ATM). Because the zone redundant configuration in the Premium or Business Critical service tiers does not create additional database redundancy, you can enable it at no extra cost. By selecting a zone redundant configuration, you can make your Premium or Business Critical databases resilient to a much larger set of failures, including catastrophic datacenter outages, without any changes to the application logic. You can also convert any existing Premium or Business Critical databases or pools to the zone redundant configuration.

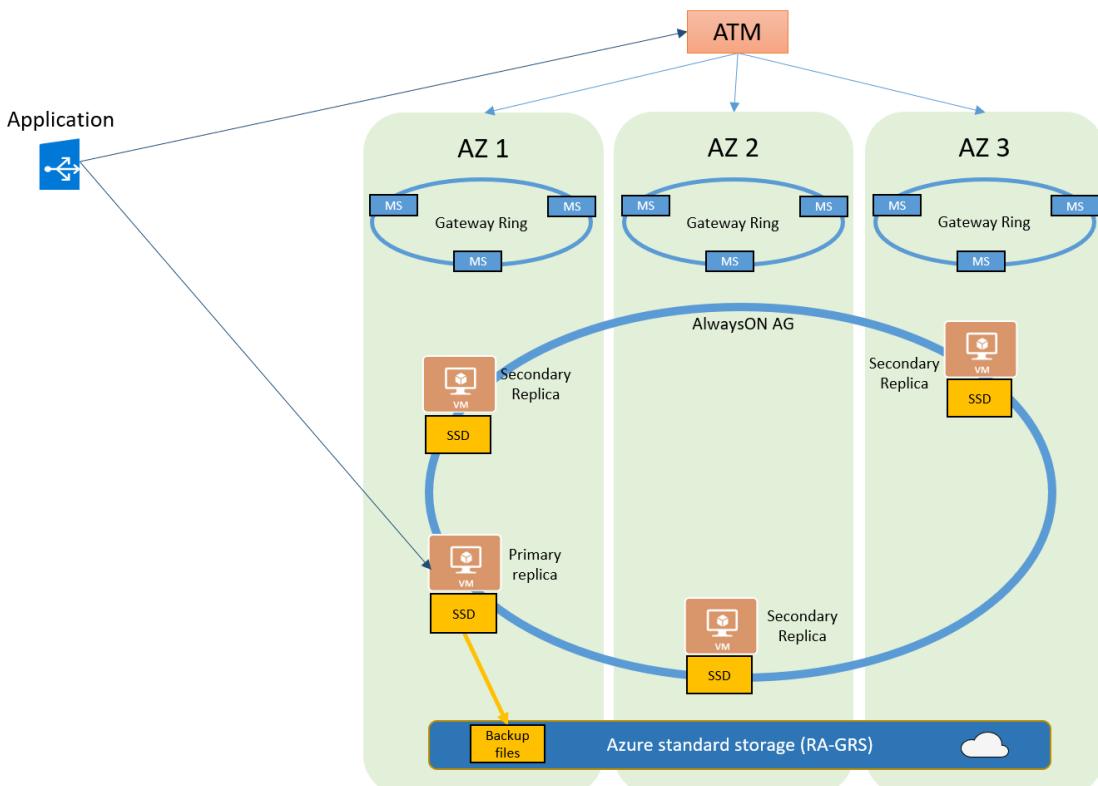
Because the zone redundant databases have replicas in different datacenters with some distance between them, the increased network latency may increase the commit time and thus impact the performance of some OLTP workloads. You can always return to the single-zone configuration by disabling the zone redundancy setting. This process is an online operation similar to the regular service tier upgrade. At the end of the process, the database or pool is migrated from a zone redundant ring to a single zone ring or vice versa.

IMPORTANT

Zone redundant databases and elastic pools are currently only supported in the Premium and Business Critical service tiers in select regions. When using the Business Critical tier, zone redundant configuration is only available when the Gen5 compute hardware is selected. For up to date information about the regions that support zone redundant databases, see [Services support by region](#).

This feature is not available in Managed instance.

The zone redundant version of the high availability architecture is illustrated by the following diagram:



Accelerated Database Recovery (ADR)

[Accelerated Database Recovery\(ADR\)](#) is a new SQL database engine feature that greatly improves database availability, especially in the presence of long running transactions. ADR is currently available for single databases, elastic pools, and Azure SQL Data Warehouse.

Testing application fault resiliency

High availability is a fundamental part of Azure SQL Database platform that works transparently for your database application. However, we recognize that you may want to test how the automatic failover operations initiated during planned or unplanned events would impact the application before you deploy it to production. You can call a special API to restart a database or an elastic pool, which will in turn trigger a failover. In the case of a zone redundant database or elastic pool, the API call would result in redirecting client connections to the new primary in an Availability Zone different from the Availability Zone of the old primary. So in addition to testing how failover impacts existing database sessions, you can also verify if it changes the end-to-end performance due to changes in network latency. Because the restart operation is intrusive and a large number of them could stress the platform, only one failover call is allowed every 30 minutes for each database or elastic pool.

A failover can be initiated using REST API or PowerShell. For REST API, see [Database failover](#) and [Elastic pool failover](#). For PowerShell, see [Invoke-AzSqlDatabaseFailover](#) and [Invoke-AzSqlElasticPoolFailover](#). The REST API calls can also be made from Azure CLI using `az rest` command.

IMPORTANT

The Failover command is currently not available in the Hyperscale service tier and for Managed Instance.

Conclusion

Azure SQL Database features a built-in high availability solution, that is deeply integrated with the Azure platform. It is dependent on Service Fabric for failure detection and recovery, on Azure Blob storage for data protection, and on Availability Zones for higher fault tolerance. In addition, Azure SQL database leverages the Always On Availability Group technology from SQL Server for replication and failover. The combination of these technologies enables applications to fully realize the benefits of a mixed storage model and support the most demanding SLAs.

Next steps

- Learn about [Azure Availability Zones](#)
- Learn about [Service Fabric](#)
- Learn about [Azure Traffic Manager](#)
- For more options for high availability and disaster recovery, see [Business Continuity](#)

Automated backups

2/18/2020 • 14 minutes to read • [Edit Online](#)

SQL Database automatically creates the database backups that are kept for the duration of the configured retention period, and uses Azure [read-access geo-redundant storage \(RA-GRS\)](#) to ensure that they are preserved even if the data center is unavailable. These backups are created automatically. Database backups are an essential part of any business continuity and disaster recovery strategy because they protect your data from accidental corruption or deletion. If your security rules require that your backups are available for an extended period of time (up to 10 years), you can configure a [long-term retention](#) on Singleton databases and Elastic pools.

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

What is a SQL Database backup

SQL Database uses SQL Server technology to create [full backups](#) every week, [differential backups](#) every 12 hours, and [transaction log backups](#) every 5-10 minutes. The backups are stored in [RA-GRS storage blobs](#) that are replicated to a [paired data center](#) for protection against a data center outage. When you restore a database, the service figures out which full, differential, and transaction log backups need to be restored.

You can use these backups to:

- **Restore an existing database to a point-in-time in the past** within the retention period using the Azure portal, Azure PowerShell, Azure CLI, or REST API. In Single database and Elastic pools, this operation will create a new database in the same server as the original database. In Managed Instance, this operation can create a copy of the database or same or different Managed Instance under the same subscription.
- **Restore a deleted database to the time it was deleted** or anytime within the retention period. The deleted database can only be restored in the same logical server or Managed Instance where the original database was created.
- **Restore a database to another geographical region.** Geo-restore allows you to recover from a geographic disaster when you cannot access your server and database. It creates a new database in any existing server anywhere in the world.
- **Restore a database from a specific long-term backup** on Single Database or Elastic Pool if the database has been configured with a long-term retention policy (LTR). LTR allows you to restore an old version of the database using the [Azure portal](#) or [Azure PowerShell](#) to satisfy a compliance request or to run an old version of the application. For more information, see [Long-term retention](#).
- To perform a restore, see [restore database from backups](#).

NOTE

In Azure storage, the term *replication* refers to copying files from one location to another. SQL's *database replication* refers to keeping multiple secondary databases synchronized with a primary database.

You can try some of these operations using the following examples:

	THE AZURE PORTAL	AZURE POWERSHELL
Change backup retention	Single Database Managed Instance	Single Database Managed Instance
Change Long-term backup retention	Single database Managed Instance - N/A	Single Database Managed Instance - N/A
Restore database from point-in-time	Single database	Single database Managed Instance
Restore deleted database	Single database	Single database Managed Instance
Restore database from Azure Blob Storage	Single database - N/A Managed Instance - N/A	Single database - N/A Managed Instance

Backup frequency

Point-in-time restore

SQL Database supports self-service for point-in-time restore (PITR) by automatically creating full backup, differential backups, and transaction log backups. Full database backups are created weekly, differential database backups are generally created every 12 hours, and transaction log backups are generally created every 5 - 10 minutes, with the frequency based on the compute size and amount of database activity. The first full backup is scheduled immediately after a database is created. It usually completes within 30 minutes, but it can take longer when the database is of a significant size. For example, the initial backup can take longer on a restored database or a database copy. After the first full backup, all further backups are scheduled automatically and managed silently in the background. The exact timing of all database backups is determined by the SQL Database service as it balances the overall system workload. You cannot change or disable the backup jobs.

The PITR backups are protected with geo-redundant storage. For more information, see [Azure Storage redundancy](#).

For more information, see [Point-in-time restore](#)

Long-term retention

Single and pooled databases offer the option of configuring long-term retention (LTR) of full backups for up to 10 years in Azure Blob storage. If LTR policy is enabled, the weekly full backups are automatically copied to a different RA-GRS storage container. To meet different compliance requirement, you can select different retention periods for weekly, monthly and/or yearly backups. The storage consumption depends on the selected frequency of backups and the retention period(s). You can use the [LTR pricing calculator](#) to estimate the cost of LTR storage.

Like PITR, the LTR backups are protected with geo-redundant storage. For more information, see [Azure Storage redundancy](#).

For more information, see [Long-term backup retention](#).

Backup storage consumption

For single databases, the total backup storage usage is calculated as follows:

Total backup storage size = (size of full backups + size of differential backups + size of log backups) - database size

For elastic pools, the total backup storage size is aggregated at the pool level and is calculated as follows:

Total backup storage size = (total size of all full backups + total size of all differential backups + total size of all log backups) - allocated pool data storage

Backups that are older than the retention period are automatically purged based on their timestamp. Because the differential backups and log backups require an earlier full backup to be useful, they are purged together in weekly chunks.

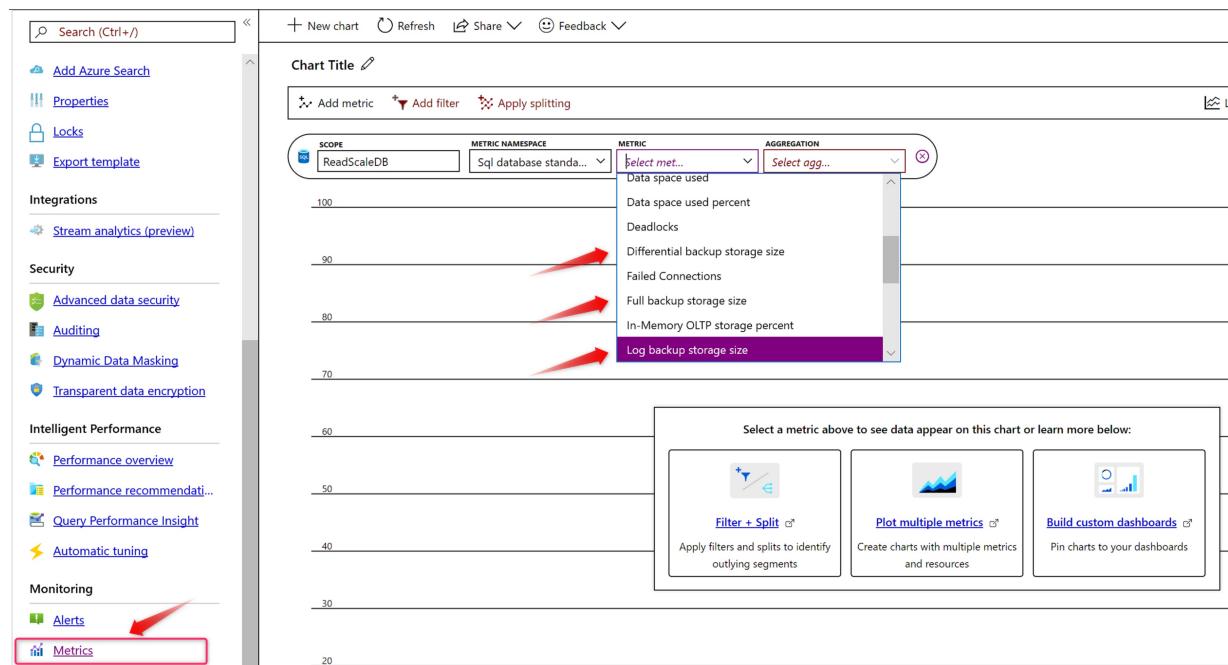
Azure SQL Database will compute your total in-retention backup storage as a cumulative value. Every hour, this value is reported to the Azure billing pipeline which is responsible for aggregating this hourly usage to calculate your consumption at the end of each month. After the database is dropped, consumption decreases as backups age. Once the backups become older than the retention period, the billing stops.

IMPORTANT

Backups of a database are retained for the specified retention period, even if the database has been dropped. While dropping and recreating a database frequently may save on storage and compute costs, it may increase backup storage costs as we retain a backup for the specified retention period (which is 7 days at minimum) for each dropped database, every time its dropped.

Monitor consumption

Each type of backup (full, differential and log) is reported on the database monitoring blade as a separate metric. The following diagram shows how to monitor the backups storage consumption for a single database. This feature is currently unavailable for managed instances.



Fine tune the backup storage consumption

The excess backup storage consumption will depend on the workload and size of the individual databases. You can consider implementing some of the following tuning techniques to further reduce your backup storage consumption:

- Reduce the [backup retention period](#) to the minimum possible for your needs.
- Avoid performing large write operations more frequently than needed, such as index rebuilds.
- For large data load operations consider using [clustered columnstore indexes](#), reduce number of non-clustered indexes, and also consider bulk load operations with row count around one million.

- In General Purpose service tier, the provisioned data storage is less expensive than the price of the excess backup storage due to which customers with continuously high excess backup storage costs may consider increasing the data storage in order to save on the backup storage.
- Use TempDB in your ETL logic for storing temporary results, instead of permanent tables (applicable to managed instance only).
- Consider turning off TDE encryption for databases that do not contain sensitive data (development or test databases, for instance). Backups for non-encrypted databases are typically compressed with a higher compression ratio.

IMPORTANT

For analytical, data mart \ data warehouse workloads it is strongly recommended to use [clustered columnstore indexes](#), reduce the number of non-clustered indexes, and also consider bulk load operations with row count around one million to reduce the excess backup storage consumption.

Storage costs

The price for storage varies if you're using the DTU model or the vCore model.

DTU Model

There is no additional charge for backup storage for databases and elastic pools using the DTU model.

vCore model

For single databases, a minimum backup storage amount equal to 100% of database size is provided at no extra charge. For elastic pools and managed instances, a minimum backup storage amount equal to 100% of the allocated data storage for the pool or instance size, respectively, is provided at no extra charge. Additional consumption of backup storage will be charged in GB/month. This additional consumption will depend on the workload and size of the individual databases.

Azure SQL DB will compute your total in-retention backup storage as a cumulative value. Every hour, this value is reported to the Azure billing pipeline which is responsible for aggregating this hourly usage to get your consumption at the end of each month. After the database is dropped, we decrease the consumption as the backups age. Once they become older than the retention period, the billing stops. Because all the log backups and differential backups are retained for the full retention period, databases that are heavily modified will have higher backup charges.

Let's assume the database has accumulated 744 GB of backup storage and this amount stays constant throughout an entire month. To convert this cumulative storage consumption to an hourly usage, we divide it by 744.0 (31 days per month * 24 hours per day). Thus, SQL DB will report the database consumed 1 GB of PITR backup each hour. Azure billing will aggregate this and show a usage of 744 GB for the entire month and the cost based on the \$/GB/mo rate in your region.

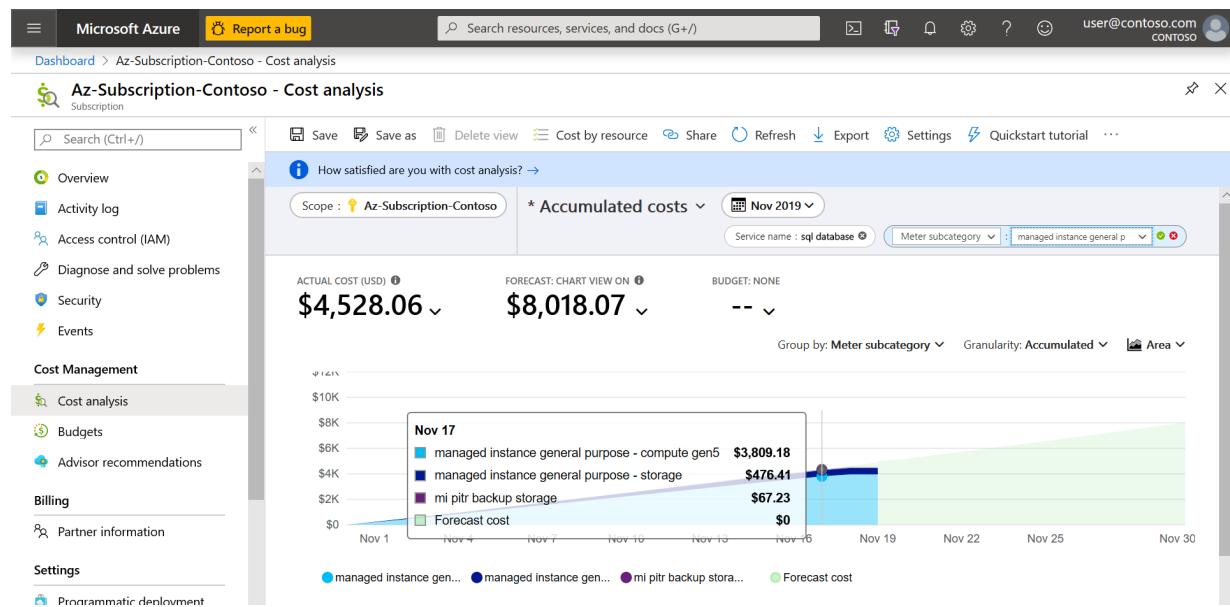
Now, a more complex example. Suppose the database has its retention increased to 14 days in the middle of the month and this (hypothetically) results in the total backup storage doubling to 1488 GB. SQL DB would report 1 GB of usage for hours 1-372, and then report the usage as 2 GB for hours 373-744. This would be aggregated to be a final bill of 1116 GB/mo.

Monitor costs

To understand the backup storage costs, go to **Cost management + Billing** from the Azure portal, select **Cost Management**, and then select **Cost analysis**. Select the desired subscription as the **Scope**, and then filter for the time period and service you're interested in.

Add a filter for **Service name**, and then choose **sql database** from the drop down. Use the **meter subcategory** filter to choose the billing counter for your service. For a single database or an elastic pool,

choose **single/elastic pool pitr backup storage**. For a managed instance, choose **mi pitr backup storage**. **Storage** and **compute** subcategories may interest you as well, though they are not associated with backup storage costs.



Backup retention

All Azure SQL databases (single, pooled, and managed instance databases) have a default backup retention period of **seven** days. You can [change backup retention period up to 35 days](#).

If you delete a database, SQL Database will keep the backups in the same way it would for an online database. For example, if you delete a Basic database that has a retention period of seven days, a backup that is four days old is saved for three more days.

If you need to keep the backups for longer than the maximum retention period, you can modify the backup properties to add one or more long-term retention periods to your database. For more information, see [Long-term retention](#).

IMPORTANT

If you delete the Azure SQL server that hosts SQL databases, all elastic pools and databases that belong to the server are also deleted and cannot be recovered. You cannot restore a deleted server. But if you configured long-term retention, the backups for the databases with LTR will not be deleted and these databases can be restored.

Encrypted backups

If your database is encrypted with TDE, the backups are automatically encrypted at rest, including LTR backups. When TDE is enabled for an Azure SQL database, backups are also encrypted. All new Azure SQL databases are configured with TDE enabled by default. For more information on TDE, see [Transparent Data Encryption with Azure SQL Database](#).

Backup integrity

On an ongoing basis, the Azure SQL Database engineering team automatically tests the restore of automated database backups of databases placed in Logical servers and Elastic pools (this is not available in Managed Instance). Upon point-in-time restore, databases also receive integrity checks using DBCC CHECKDB.

Managed Instance takes automatic initial backup with `CHECKSUM` of the databases restored using native `RESTORE` command or Data Migration Service once the migration is completed.

Any issues found during the integrity check will result in an alert to the engineering team. For more information about data integrity in Azure SQL Database, see [Data Integrity in Azure SQL Database](#).

Compliance

When you migrate your database from a DTU-based service tier to a vCore-based service tier, the PITR retention is preserved to ensure that your application's data recovery policy is not compromised. If the default retention doesn't meet your compliance requirements, you can change the PITR retention period using PowerShell or REST API. For more information, see [Change Backup Retention Period](#).

NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Change PITR backup retention period

You can change the default PITR backup retention period using the Azure portal, PowerShell, or the REST API. The following examples illustrate how to change PITR retention to 28 days.

WARNING

If you reduce the current retention period, all existing backups older than the new retention period are no longer available. If you increase the current retention period, SQL Database will keep the existing backups until the longer retention period is reached.

NOTE

These APIs will only impact the PITR retention period. If you configured LTR for your database, it will not be impacted. For more information about how to change the LTR retention period(s), see [Long-term retention](#).

Change PITR backup retention period using Azure portal

To change the PITR backup retention period using the Azure portal, navigate to the server object whose retention period you wish to change within the portal and then select the appropriate option based on which server object you're modifying.

- [Single database & Elastic pools](#)
- [Managed Instance](#)

Change of PITR backup retention for single Azure SQL Databases is performed at the server level. Change made at the server level applies to databases on that server. To change PITR for Azure SQL Database server from Azure portal, navigate to the server overview blade, click on Manage Backups on the navigation menu, and then click on Configure retention at the navigation bar.

The screenshot shows the Azure portal interface for managing backups. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Quick start, Failover groups, Manage Backups), Security (Advanced Data Security, Auditing), and SQL elastic pools. The main area displays a table of databases with columns: DATABASE, PITR BACKUPS, and WEEKLY LTR. Two databases are listed: basetenantdb (28 days, 52 Weeks) and tenantcatalog (28 days, 52 Weeks). A modal window titled 'Configure policies' is open, specifically the 'Point In Time Restore Configuration' section. It shows settings for 'Configure PtTR backup retention' (Keep current settings, Days: 7, 14, 21, 28, 35, Week(s)), 'Monthly LTR Backups' (2 weeks), and 'Yearly LTR Backups' (Week 15, 10 weeks).

Change PITR backup retention period using PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

```
Set-AzSqlDatabaseBackupShortTermRetentionPolicy -ResourceGroupName resourceGroup -ServerName testserver -  
DatabaseName testDatabase -RetentionDays 28
```

Change PITR retention period using REST API

Sample Request

```
PUT https://management.azure.com/subscriptions/00000000-1111-2222-3333-  
444444444444/resourceGroups/resourceGroup/providers/Microsoft.Sql/servers/testserver/databases/testDatabase  
e/backupShortTermRetentionPolicies/default?api-version=2017-10-01-preview
```

Request Body

```
{
  "properties": {
    "retentionDays": 28
  }
}
```

Sample Response

Status code: 200

```
{  
  "id": "/subscriptions/00000000-1111-2222-3333-  
444444444444/providers/Microsoft.Sql/resourceGroups/resourceGroup/servers/testserver/databases/testDatabase  
e/backupShortTermRetentionPolicies/default",  
  "name": "default",  
  "type": "Microsoft.Sql/resourceGroups/servers/databases/backupShortTermRetentionPolicies",  
  "properties": {  
    "retentionDays": 28  
  }  
}
```

For more information, see [Backup Retention REST API](#).

Next steps

- Database backups are an essential part of any business continuity and disaster recovery strategy because they protect your data from accidental corruption or deletion. To learn about the other Azure SQL Database business continuity solutions, see [Business continuity overview](#).
- To restore to a point in time using the Azure portal, see [restore database to a point in time using the Azure portal](#).
- To restore to a point in time using PowerShell, see [restore database to a point in time using PowerShell](#).
- To configure, manage, and restore from long-term retention of automated backups in Azure Blob storage using the Azure portal, see [Manage long-term backup retention using the Azure portal](#).
- To configure, manage, and restore from long-term retention of automated backups in Azure Blob storage using PowerShell, see [Manage long-term backup retention using PowerShell](#).

Recover an Azure SQL database by using automated database backups

11/19/2019 • 10 minutes to read • [Edit Online](#)

By default, Azure SQL Database backups are stored in geo-replicated blob storage (RA-GRS storage type). The following options are available for database recovery by using [automated database backups](#). You can:

- Create a new database on the same SQL Database server, recovered to a specified point in time within the retention period.
- Create a database on the same SQL Database server, recovered to the deletion time for a deleted database.
- Create a new database on any SQL Database server in the same region, recovered to the point of the most recent backups.
- Create a new database on any SQL Database server in any other region, recovered to the point of the most recent replicated backups.

If you configured [backup long-term retention](#), you can also create a new database from any long-term retention backup on any SQL Database server.

IMPORTANT

You can't overwrite an existing database during restore.

When you're using the Standard or Premium service tiers, your database restore might incur an extra storage cost. The extra cost is incurred when the maximum size of the restored database is greater than the amount of storage included with the target database's service tier and performance level. For pricing details of extra storage, see the [SQL Database pricing page](#). If the actual amount of used space is less than the amount of storage included, you can avoid this extra cost by setting the maximum database size to the included amount.

Recovery time

The recovery time to restore a database by using automated database backups is affected by several factors:

- The size of the database.
- The compute size of the database.
- The number of transaction logs involved.
- The amount of activity that needs to be replayed to recover to the restore point.
- The network bandwidth if the restore is to a different region.
- The number of concurrent restore requests being processed in the target region.

For a large or very active database, the restore might take several hours. If there is a prolonged outage in a region, it's possible that a high number of geo-restore requests will be initiated for disaster recovery. When there are many requests, the recovery time for individual databases can increase. Most database restores complete in less than 12 hours.

For a single subscription, there are limitations on the number of concurrent restore requests. These limitations apply to any combination of point-in-time restores, geo-restores, and restores from long-term retention backup.

	MAX # OF CONCURRENT REQUESTS BEING PROCESSED	MAX # OF CONCURRENT REQUESTS BEING SUBMITTED
Single database (per subscription)	10	60
Elastic pool (per pool)	4	200

There isn't a built-in method to restore the entire server. For an example of how to accomplish this task, see [Azure SQL Database: Full Server Recovery](#).

IMPORTANT

To recover by using automated backups, you must be a member of the SQL Server contributor role in the subscription, or be the subscription owner. For more information, see [RBAC: Built-in roles](#). You can recover by using the Azure portal, PowerShell, or the REST API. You can't use Transact-SQL.

Point-in-time restore

You can restore a standalone, pooled, or instance database to an earlier point in time by using the Azure portal, [PowerShell](#), or the [REST API](#). The request can specify any service tier or compute size for the restored database. Ensure that you have sufficient resources on the server to which you are restoring the database. When complete, the restore creates a new database on the same server as the original database. The restored database is charged at normal rates, based on its service tier and compute size. You don't incur charges until the database restore is complete.

You generally restore a database to an earlier point for recovery purposes. You can treat the restored database as a replacement for the original database, or use it as a data source to update the original database.

- **Database replacement**

If you intend the restored database to be a replacement for the original database, you should specify the original database's compute size and service tier. You can then rename the original database, and give the restored database the original name by using the [ALTER DATABASE](#) command in T-SQL.

- **Data recovery**

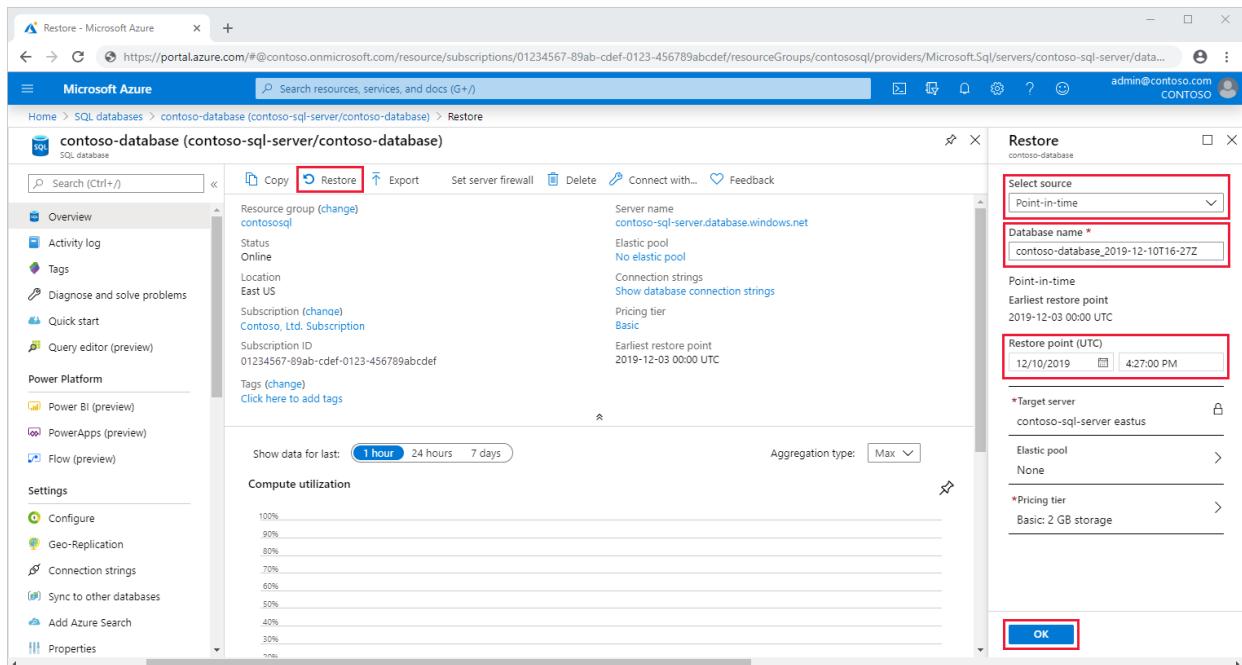
If you plan to retrieve data from the restored database to recover from a user or application error, you need to write and execute a data recovery script that extracts data from the restored database and applies to the original database. Although the restore operation may take a long time to complete, the restoring database is visible in the database list throughout the restore process. If you delete the database during the restore, the restore operation will be canceled and you will not be charged for the database that did not complete the restore.

Point-in-time restore by using Azure portal

You can recover a single SQL database or instance database to a point in time from the overview blade of the database you want to restore in the Azure portal.

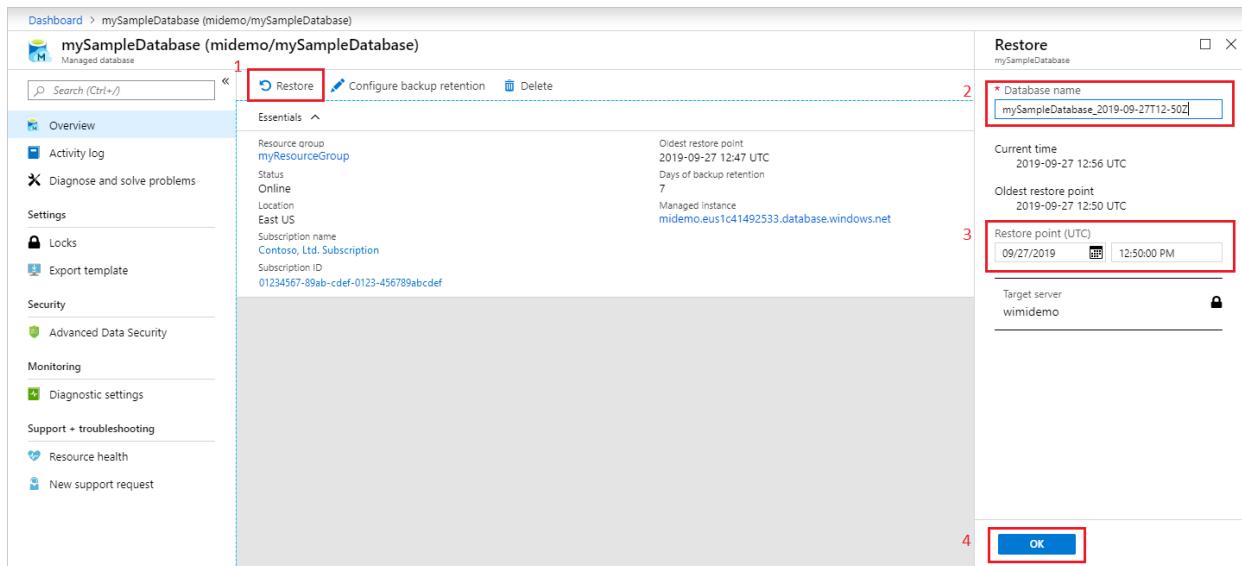
Single Azure SQL database

To recover a single or pooled database to a point in time by using the Azure portal, open the database overview page, and select **Restore** on the toolbar. Choose the backup source, and select the point-in-time backup point from which a new database will be created.



Managed instance database

To recover a managed instance database to a point in time by using the Azure portal, open the database overview page, and select **Restore** on the toolbar. Choose the point-in-time backup point from which a new database will be created.



TIP

To programmatically restore a database from a backup, see [Programmatically performing recovery using automated backups](#).

Deleted database restore

You can restore a deleted database to the deletion time, or an earlier point in time, on the same SQL Database server or the same managed instance. You can accomplish this through the Azure portal, [PowerShell](#), or the [REST \(createMode=Restore\)](#). You restore a deleted database by creating a new database from the backup.

IMPORTANT

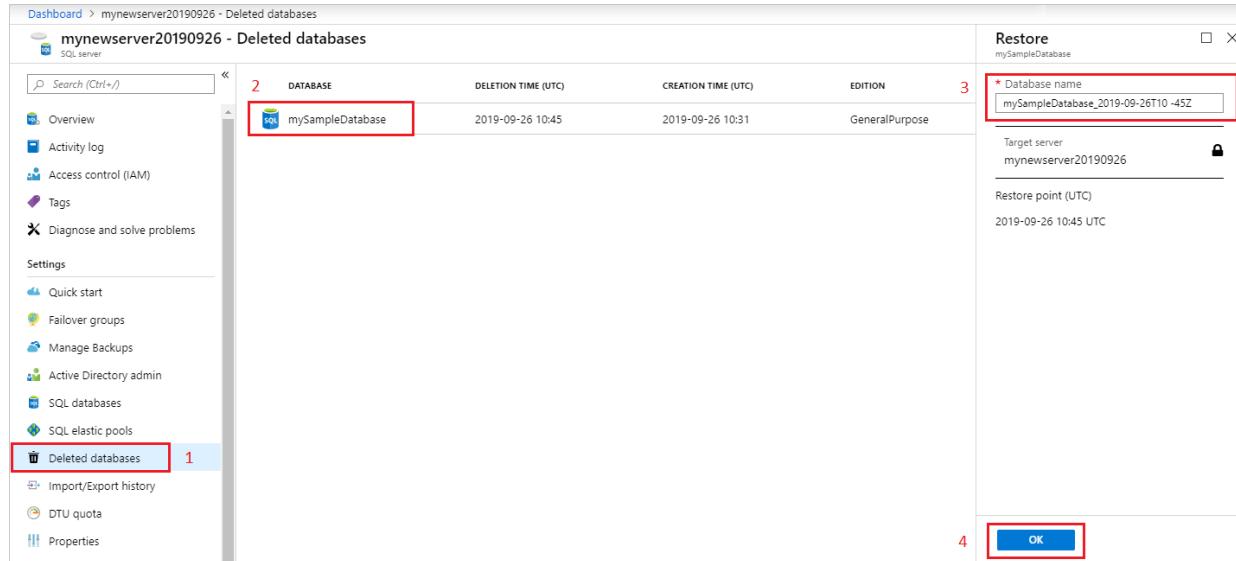
If you delete an Azure SQL Database server or managed instance, all its databases are also deleted, and can't be recovered. You can't restore a deleted server or managed instance.

Deleted database restore by using the Azure portal

You restore deleted databases from the Azure portal from the server and instance resource.

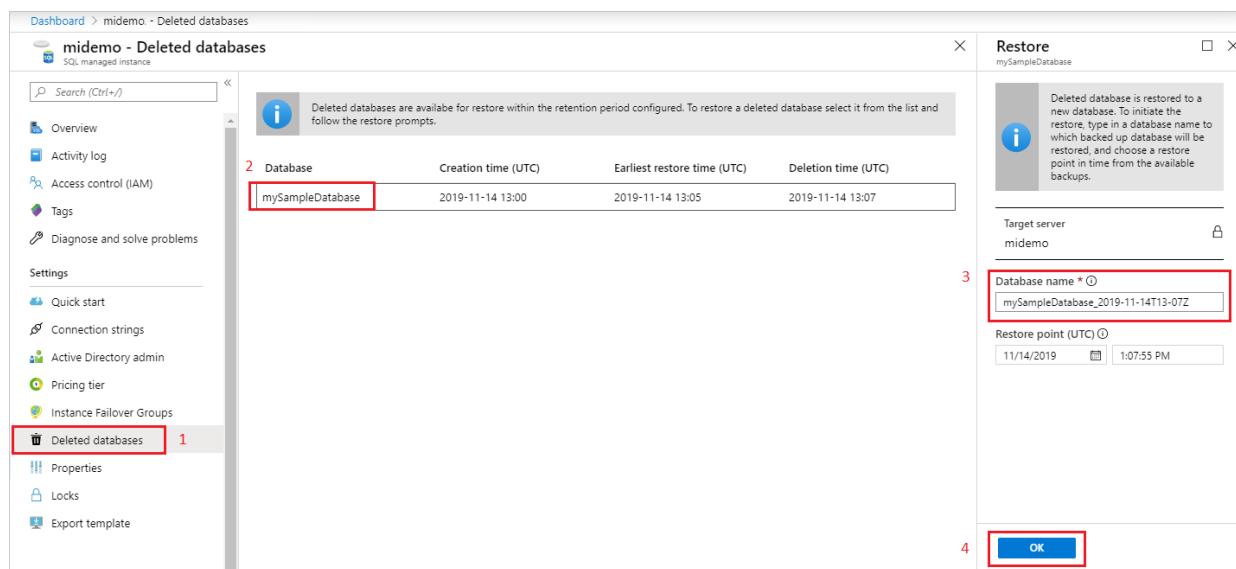
Single Azure SQL database

To recover a single or pooled deleted database to the deletion time by using the Azure portal, open the server overview page, and select **Deleted databases**. Select a deleted database that you want to restore, and type the name for the new database that will be created with data restored from the backup.



Managed instance database

To recover a managed database by using the Azure portal, open the managed instance overview page, and select **Deleted databases**. Select a deleted database that you want to restore, and type the name for the new database that will be created with data restored from the backup.



Deleted database restore by using PowerShell

Use the following sample scripts to restore a deleted database for Azure SQL Database and a managed instance by using PowerShell.

Single Azure SQL database

For a sample PowerShell script showing how to restore a deleted Azure SQL database, see [Restore a SQL](#)

database using PowerShell.

Managed instance database

For a sample PowerShell script showing how to restore a deleted instance database, see [Restore deleted database on managed instance using PowerShell](#).

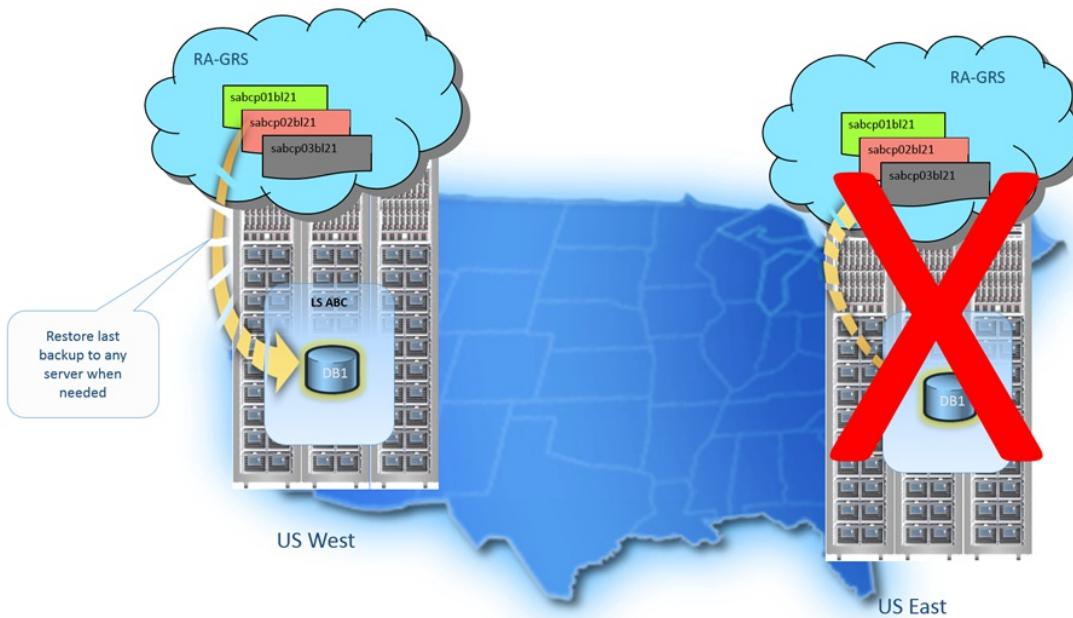
TIP

To programmatically restore a deleted database, see [Programmatically performing recovery using automated backups](#).

Geo-restore

You can restore a SQL database on any server in any Azure region from the most recent geo-replicated backups. Geo-restore uses a geo-replicated backup as its source. You can request geo-restore even if the database or datacenter is inaccessible due to an outage.

Geo-restore is the default recovery option when your database is unavailable because of an incident in the hosting region. You can restore the database to a server in any other region. There is a delay between when a backup is taken and when it is geo-replicated to an Azure blob in a different region. As a result, the restored database can be up to one hour behind the original database. The following illustration shows a database restore from the last available backup in another region.



Geo-restore by using the Azure portal

From the Azure portal, you create a new single or managed instance database, and select an available geo-restore backup. The newly created database contains the geo-restored backup data.

Single Azure SQL database

To geo-restore a single SQL database from the Azure portal in the region and server of your choice, follow these steps:

1. From **Dashboard**, select **Add > Create SQL Database**. On the **Basics** tab, enter the required information.
2. Select **Additional settings**.
3. For **Use existing data**, select **Backup**.

4. For **Backup**, select a backup from the list of available geo-restore backups.

1 Create SQL Database Microsoft

2 Basics Additional settings Tags Review + create

Customize additional configuration parameters including collation & sample data.

Data source

Start with a blank database, restore from a backup or select sample data to populate your new database.

* Use existing data **3 None Backup Sample**

* Backup **4**

You can also restore a database to a server blade. [Learn more](#)

Database Collation

Database collation defines the rules that sort and compare data, and cannot be changed after database creation. The default database collation is SQL_Latin1_General_CI_AS. [Learn more](#)

Complete the process of creating a new database from the backup. When you create the single Azure SQL database, it contains the restored geo-restore backup.

Managed instance database

To geo-restore a managed instance database from the Azure portal to an existing managed instance in a region of your choice, select a managed instance on which you want a database to be restored. Follow these steps:

1. Select **New database**.
2. Type a desired database name.
3. Under **Use existing data**, select **Backup**.
4. Select a backup from the list of available geo-restore backups.

Dashboard > wimidemo

midemo SQL managed instance

1 **+ New database** **2** **Reset password** **Delete** **Feedback**

Your Managed Instance is ready. Click here to get started.

Resource group (change) : myRG Managed instance

Status : Online Host

Location : East US Pricing tier

Subscription (change) : Dev/test subscription Virtual network/

Subscription ID : 75abcd-abcd-77dd-91cc-3e4f21ca Virtual cluster

Tags (change) : Retain : yes

CPU utilization

1 hour 24 hours 7 days View: Avg

Managed database

* Database name **Enter database name**

Managed instance **wimidemo**

* Use existing data **None Backup**

* Backup **4**

Select a backup

myserver (West Europe)

- database1 (2019-09-16 (12:05:30 UTC))
- database2 (2019-09-16 (12:06:45 UTC))
- database3 (2019-09-16 (12:07:51 UTC))
- database4 (2019-09-16 (12:08:38 UTC))
- database5 (2019-09-16 (12:09:23 UTC))
- database6 (2019-09-16 (12:10:41 UTC))
- database7 (2019-09-16 (12:11:38 UTC))

Complete the process of creating a new database. When you create the instance database, it contains the restored geo-restore backup.

Geo-restore by using PowerShell

Single Azure SQL database

For a PowerShell script that shows how to perform geo-restore for a single SQL database, see [Use PowerShell to restore an Azure SQL single database to an earlier point in time](#).

Managed instance database

For a PowerShell script that shows how to perform geo-restore for a managed instance database, see [Use PowerShell to restore a managed instance database to another geo-region](#).

Geo-restore considerations

You can't perform a point-in-time restore on a geo-secondary database. You can only do so on a primary database. For detailed information about using geo-restore to recover from an outage, see [Recover from an outage](#).

IMPORTANT

Geo-restore is the most basic disaster recovery solution available in SQL Database. It relies on automatically created geo-replicated backups with recovery point objective (RPO) equal to 1 hour, and the estimated recovery time of up to 12 hours. It doesn't guarantee that the target region will have the capacity to restore your databases after a regional outage, because a sharp increase of demand is likely. If your application uses relatively small databases and is not critical to the business, geo-restore is an appropriate disaster recovery solution. For business-critical applications that require large databases and must ensure business continuity, use [Auto-failover groups](#). It offers a much lower RPO and recovery time objective, and the capacity is always guaranteed. For more information on business continuity choices, see [Overview of business continuity](#).

Programmatically performing recovery by using automated backups

You can also use Azure PowerShell or the REST API for recovery. The following tables describe the set of commands available.

PowerShell

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). Arguments for the commands in the Az module and in AzureRm modules are to a great extent identical.

Single Azure SQL database

To restore a standalone or pooled database, see [Restore-AzSqlDatabase](#).

CMDLET	DESCRIPTION
Get-AzSqlDatabase	Gets one or more databases.
Get-AzSqlDeletedDatabaseBackup	Gets a deleted database that you can restore.
Get-AzSqlDatabaseGeoBackup	Gets a geo-redundant backup of a database.
Restore-AzSqlDatabase	Restores a SQL database.

TIP

For a sample PowerShell script that shows how to perform a point-in-time restore of a database, see [Restore a SQL database using PowerShell](#).

Managed instance database

To restore a managed instance database, see [Restore-AzSqlInstanceDatabase](#).

CMDLET	DESCRIPTION
Get-AzSqlInstance	Gets one or more managed instances.
Get-AzSqlInstanceDatabase	Gets an instance database.
Restore-AzSqlInstanceDatabase	Restores an instance database.

REST API

To restore a single or pooled database by using the REST API:

API	DESCRIPTION
REST (createMode=Recovery)	Restores a database.
Get Create or Update Database Status	Returns the status during a restore operation.

Azure CLI

Single Azure SQL database

To restore a single or pooled database by using the Azure CLI, see [az sql db restore](#).

Managed instance database

To restore a managed instance database by using the Azure CLI, see [az sql midb restore](#).

Summary

Automatic backups protect your databases from user and application errors, accidental database deletion, and prolonged outages. This built-in capability is available for all service tiers and compute sizes.

Next steps

- [Business continuity overview](#)
- [SQL Database automated backups](#)
- [Long-term retention](#)
- To learn about faster recovery options, see [Active geo-replication](#) or [Auto-failover groups](#).

Accelerated Database Recovery

11/7/2019 • 5 minutes to read • [Edit Online](#)

Accelerated Database Recovery(ADR) is a new SQL database engine feature that greatly improves database availability, especially in the presence of long running transactions, by redesigning the SQL database engine recovery process. ADR is currently available for single databases and pooled databases in Azure SQL Database, and databases in Azure SQL Data Warehouse (currently in public preview). The primary benefits of ADR are:

- **Fast and consistent database recovery**

With ADR, long running transactions do not impact the overall recovery time, enabling fast and consistent database recovery irrespective of the number of active transactions in the system or their sizes.

- **Instantaneous transaction rollback**

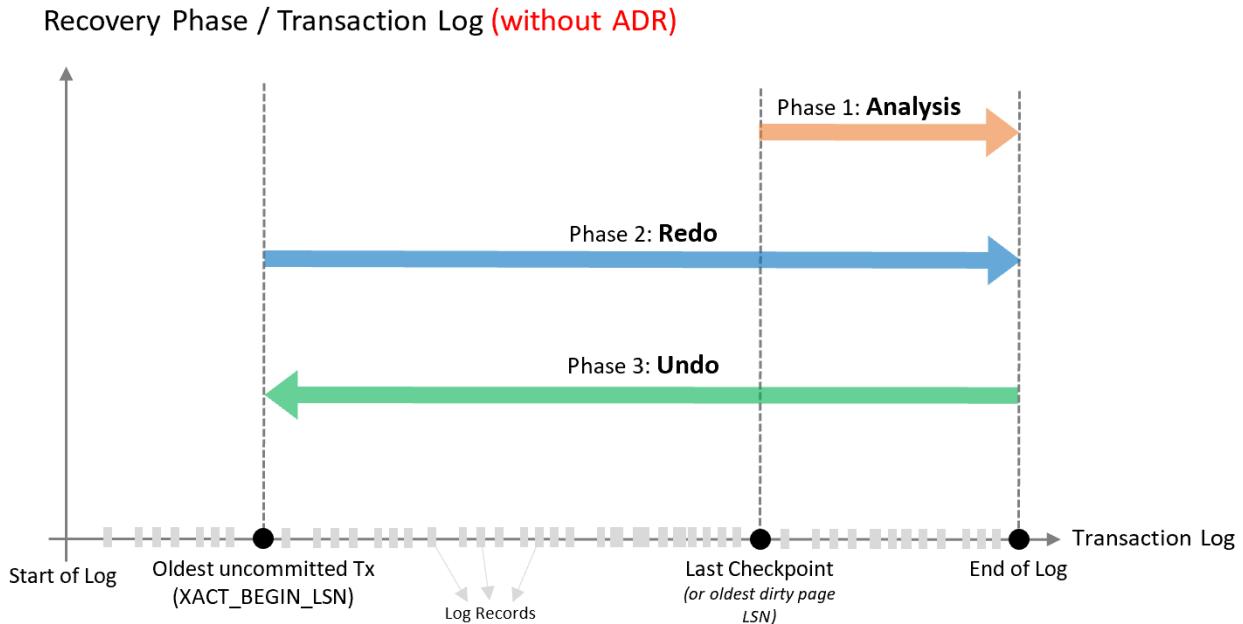
With ADR, transaction rollback is instantaneous, irrespective of the time that the transaction has been active or the number of updates that has performed.

- **Aggressive log truncation**

With ADR, the transaction log is aggressively truncated, even in the presence of active long running transactions, which prevents it from growing out of control.

The current database recovery process

Database recovery in SQL Server follows the [ARIES](#) recovery model and consists of three phases, which are illustrated in the following diagram and explained in more detail following the diagram.



- **Analysis phase**

Forward scan of the transaction log from the beginning of the last successful checkpoint (or the oldest dirty page LSN) until the end, to determine the state of each transaction at the time SQL Server stopped.

- **Redo phase**

Forward scan of the transaction log from the oldest uncommitted transaction until the end, to bring the

database to the state it was at the time of the crash by redoing all committed operations.

- **Undo phase**

For each transaction that was active as of the time of the crash, traverses the log backwards, undoing the operations that this transaction performed.

Based on this design, the time it takes the SQL database engine to recover from an unexpected restart is (roughly) proportional to the size of the longest active transaction in the system at the time of the crash. Recovery requires a rollback of all incomplete transactions. The length of time required is proportional to the work that the transaction has performed and the time it has been active. Therefore, the SQL Server recovery process can take a long time in the presence of long running transactions (such as large bulk insert operations or index build operations against a large table).

Also, cancelling/rolling back a large transaction based on this design can also take a long time as it is using the same Undo recovery phase as described above.

In addition, the SQL database engine cannot truncate the transaction log when there are long running transactions because their corresponding log records are needed for the recovery and rollback processes. As a result of this design of the SQL database engine, some customers face the problem that the size of the transaction log grows very large and consumes huge amounts of drive space.

The Accelerated Database Recovery process

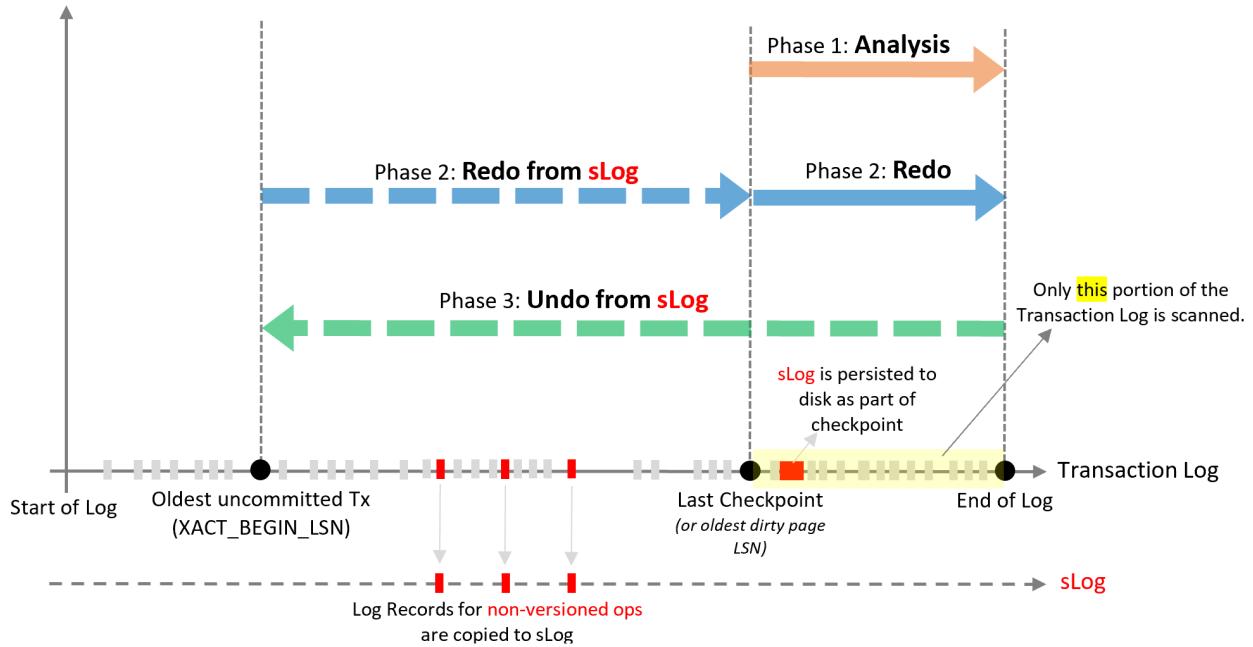
ADR addresses the above issues by completely redesigning the SQL database engine recovery process to:

- Make it constant time/instant by avoiding having to scan the log from/to the beginning of the oldest active transaction. With ADR, the transaction log is only processed from the last successful checkpoint (or oldest dirty page Log Sequence Number (LSN)). As a result, recovery time is not impacted by long running transactions.
- Minimize the required transaction log space since there is no longer a need to process the log for the whole transaction. As a result, the transaction log can be truncated aggressively as checkpoints and backups occur.

At a High Level, ADR achieves fast database recovery by versioning all physical database modifications and only undoing logical operations, which are limited and can be undone almost instantly. Any transaction that was active as of the time of a crash are marked as aborted and, therefore, any versions generated by these transactions can be ignored by concurrent user queries.

The ADR recovery process has the same three phases as the current recovery process. How these phases operate with ADR is illustrated in the following diagram and explained in more detail following the diagram.

Recovery Phase / Transaction Log / sLog (with ADR)



• Analysis phase

The process remains the same as today with the addition of reconstructing sLog and copying log records for non-versioned operations.

• Redo phase

Broken into two phases (P)

- Phase 1

Redo from sLog (oldest uncommitted transaction up to last checkpoint). Redo is a fast operation as it only needs to process a few records from the sLog.

- Phase 2

Redo from Transaction Log starts from last checkpoint (instead of oldest uncommitted transaction)

• Undo phase

The Undo phase with ADR completes almost instantaneously by using sLog to undo non-versioned operations and Persisted Version Store (PVS) with Logical Revert to perform row level version-based Undo.

ADR recovery components

The four key components of ADR are:

• Persisted Version Store (PVS)

The persisted version store is a new SQL database engine mechanism for persisting the row versions generated in the database itself instead of the traditional `tempdb` version store. PVS enables resource isolation as well as improves availability of readable secondaries.

• Logical Revert

Logical revert is the asynchronous process responsible for performing row-level version-based Undo - providing instant transaction rollback and undo for all versioned operations. Logical revert is accomplished by:

- Keeping track of all aborted transactions and marking them invisible to other transactions.
- Performing rollback by using PVS for all user transactions, rather than physically scanning the transaction log and undoing changes one at a time.
- Releasing all locks immediately after transaction abort. Since abort involves simply marking changes in memory, the process is very efficient and therefore locks do not have to be held for a long time.

- **sLog**

sLog is a secondary in-memory log stream that stores log records for non-versioned operations (such as metadata cache invalidation, lock acquisitions, and so on). The sLog is:

- Low volume and in-memory
- Persisted on disk by being serialized during the checkpoint process
- Periodically truncated as transactions commit
- Accelerates redo and undo by processing only the non-versioned operations
- Enables aggressive transaction log truncation by preserving only the required log records

- **Cleaner**

The cleaner is the asynchronous process that wakes up periodically and cleans page versions that are not needed.

Who should consider Accelerated Database Recovery

The following types of customers should consider enabling ADR:

- Customers that have workloads with long running transactions.
- Customers that have seen cases where active transactions are causing the transaction log to grow significantly.
- Customers that have experienced long periods of database unavailability due to SQL Server long running recovery (such as unexpected SQL Server restart or manual transaction rollback).

Store Azure SQL Database backups for up to 10 years

2/20/2020 • 4 minutes to read • [Edit Online](#)

Many applications have regulatory, compliance, or other business purposes that require you to retain database backups beyond the 7-35 days provided by Azure SQL Database [automatic backups](#). By using the long-term retention (LTR) feature, you can store specified SQL database full backups in Azure Blob storage with read-access geo-redundant storage for up to 10 years. You can then restore any backup as a new database. For more information about Azure Storage redundancy, see [Azure Storage redundancy](#).

NOTE

LTR can be enabled for single and pooled databases. It is not yet available for instance databases in Managed Instances. You can use SQL Agent jobs to schedule [copy-only database backups](#) as an alternative to LTR beyond 35 days.

How SQL Database long-term retention works

Long-term backup retention (LTR) leverages the full database backups that are [automatically created](#) to enable point-time restore (PITR). If an LTR policy is configured, these backups are copied to different blobs for long-term storage. The copy is a background job that has no performance impact on the database workload. The LTR policy for each SQL database can also specify how frequently the LTR backups are created.

To enable LTR, you can define a policy using a combination of four parameters: weekly backup retention (W), monthly backup retention (M), yearly backup retention (Y), and week of year (WeekOfYear). If you specify W, one backup every week will be copied to the long-term storage. If you specify M, the first backup of each month will be copied to the long-term storage. If you specify Y, one backup during the week specified by WeekOfYear will be copied to the long-term storage. If the specified WeekOfYear is in the past when the policy is configured, the first LTR backup will be created in the following year. Each backup will be kept in the long-term storage according to the policy parameters that are configured when the LTR backup is created.

NOTE

Any change to the LTR policy applies only to future backups. For example, if weekly backup retention (W), monthly backup retention (M), or yearly backup retention (Y) is modified, the new retention setting will only apply to new backups. The retention of existing backups will not be modified. If your intention is to delete old LTR backups before their retention period expires, you will need to [manually delete the backups](#).

Examples of the LTR policy:

- W=0, M=0, Y=5, WeekOfYear=3

The third full backup of each year will be kept for five years.

- W=0, M=3, Y=0

The first full backup of each month will be kept for three months.

- W=12, M=0, Y=0

Each weekly full backup will be kept for 12 weeks.

- W=6, M=12, Y=10, WeekOfYear=16

Each weekly full backup will be kept for six weeks. Except first full backup of each month, which will be kept for 12 months. Except the full backup taken on 16th week of year, which will be kept for 10 years.

The following table illustrates the cadence and expiration of the long-term backups for the following policy:

W=12 weeks (84 days), M=12 months (365 days), Y=10 years (3650 days), WeekOfYear=15 (week after April 15)

PITR backup copied to LTR	Expiration W	Expiration M	Expiration Y
3/7/2018		3/7/2019	
3/14/2018	6/6/2018		
3/21/2018	6/13/2018		
3/28/2018	6/20/2018		
4/4/2018		4/25/2019	
4/11/2018	7/4/2018		
4/18/2018	7/11/2018		
4/25/2018	7/18/2018		
5/2/2018		5/23/2019	
5/9/2018	8/1/2018		
5/16/2018			5/13/2028
5/23/2018	8/15/2018		
5/30/2018	8/22/2018		
6/6/2018		6/20/2019	
6/13/2018	9/5/2018		
6/20/2018	9/12/2018		
6/27/2018	9/19/2018		
7/4/2018		7/25/2019	
7/11/2018	10/3/2018		
7/18/2018	10/10/2018		
7/25/2018	10/17/2018		
8/1/2018		8/22/2019	
8/8/2018	10/31/2018		
8/15/2018	11/7/2018		
8/22/2018	11/14/2018		
8/29/2018	11/21/2018		

If you modify the above policy and set W=0 (no weekly backups), the cadence of backup copies will change as shown in the above table by the highlighted dates. The storage amount needed to keep these backups would reduce accordingly.

IMPORTANT

The timing of the individual LTR backups is controlled by Azure SQL Database. You cannot manually create a LTR backup or control the timing of the backup creation. After configuring an LTR policy, it may take up to 7 days before the first LTR backup will show up on the list of available backups.

Geo-replication and long-term backup retention

If you are using active geo-replication or failover groups as your business continuity solution, you should prepare for eventual failovers and configure the same LTR policy on the geo-secondary database. Your LTR storage cost will not increase as backups are not generated from the secondaries. Only when the secondary becomes primary the backups will be created. It ensures non-interrupted generation of the LTR backups when the failover is triggered and the primary moves to the secondary region.

NOTE

When the original primary database recovers from an outage that caused the failover, it will become a new secondary. Therefore, the backup creation will not resume and the existing LTR policy will not take effect until it becomes the primary again.

Configure long-term backup retention

To learn how to configure long-term retention using the Azure portal or PowerShell, see [Manage Azure SQL Database long-term backup retention](#).

Restore database from LTR backup

To restore a database from the LTR storage, you can select a specific backup based on its timestamp. The database can be restored to any existing server under the same subscription as the original database. To learn how to restore your database from an LTR backup, using the Azure portal, or PowerShell, see [Manage Azure SQL Database long-term backup retention](#).

Next steps

Because database backups protect data from accidental corruption or deletion, they're an essential part of any business continuity and disaster recovery strategy. To learn about the other SQL Database business-continuity solutions, see [Business continuity overview](#).

Creating and using active geo-replication

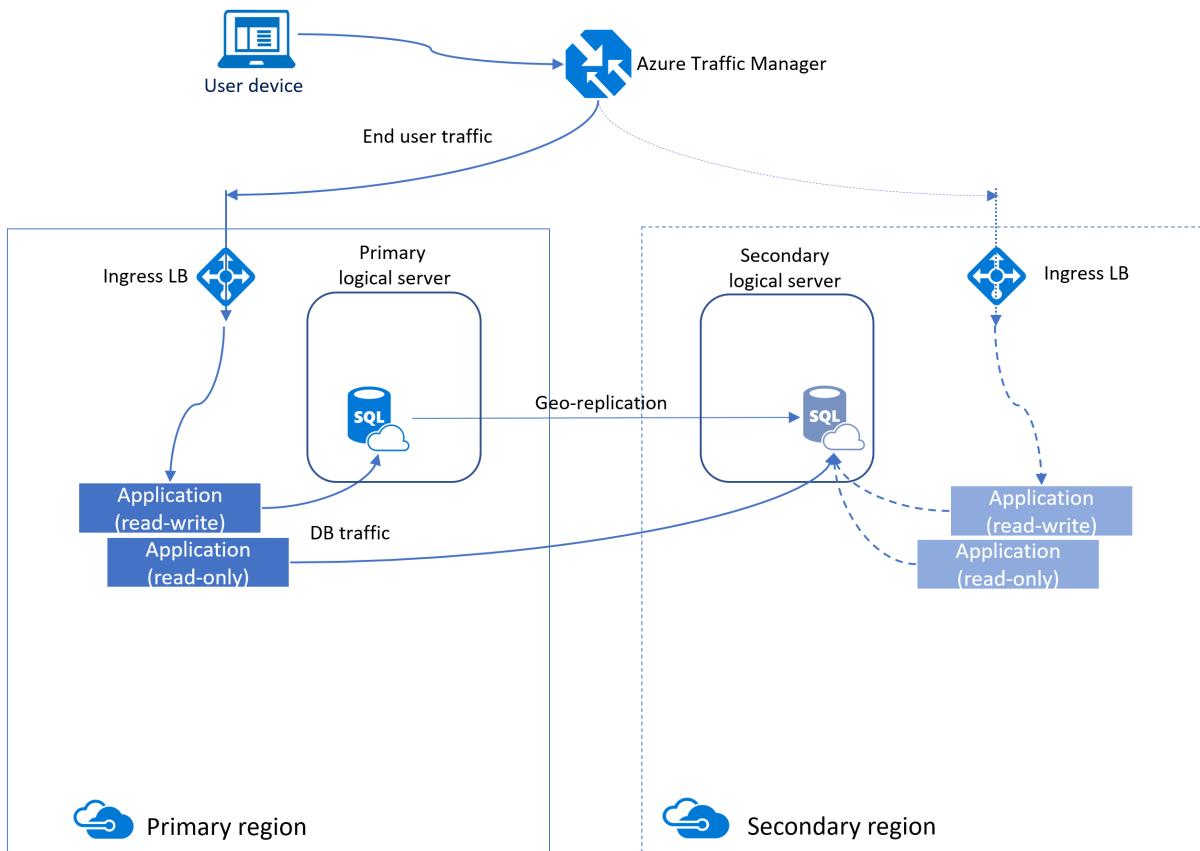
2/17/2020 • 19 minutes to read • [Edit Online](#)

Active geo-replication is an Azure SQL Database feature that allows you to create readable secondary databases of individual databases on a SQL Database server in the same or different data center (region).

NOTE

Active geo-replication is not supported by managed instance. For geographic failover of managed instances, use [Auto-failover groups](#).

Active geo-replication is designed as a business continuity solution that allows the application to perform quick disaster recovery of individual databases in case of a regional disaster or large scale outage. If geo-replication is enabled, the application can initiate failover to a secondary database in a different Azure region. Up to four secondaries are supported in the same or different regions, and the secondaries can also be used for read-only access queries. The failover must be initiated manually by the application or the user. After failover, the new primary has a different connection end point. The following diagram illustrates a typical configuration of a geo-redundant cloud application using Active geo-replication.



IMPORTANT

SQL Database also supports auto-failover groups. For more information, see using [auto-failover groups](#). Also, active geo-replication is not supported for databases created within a Managed Instance. Consider using [failover groups](#) with Managed Instances.

If for any reason your primary database fails, or simply needs to be taken offline, you can initiate failover to

any of your secondary databases. When failover is activated to one of the secondary databases, all other secondaries are automatically linked to the new primary.

You can manage replication and failover of an individual database or a set of databases on a server or in an elastic pool using active geo-replication. You can do that using:

- The [Azure portal](#)
- [PowerShell: Single database](#)
- [PowerShell: Elastic pool](#)
- [Transact-SQL: Single database or elastic pool](#)
- [REST API: Single database](#)

Active geo-replication leverages the [Always On](#) technology of SQL Server to asynchronously replicate committed transactions on the primary database to a secondary database using snapshot isolation. Auto-failover groups provide the group semantics on top of active geo-replication but the same asynchronous replication mechanism is used. While at any given point, the secondary database might be slightly behind the primary database, the secondary data is guaranteed to never have partial transactions. Cross-region redundancy enables applications to quickly recover from a permanent loss of an entire datacenter or parts of a datacenter caused by natural disasters, catastrophic human errors, or malicious acts. The specific RPO data can be found at [Overview of Business Continuity](#).

NOTE

If there is a network failure between two regions, we retry every 10 seconds to re-establish connections.

IMPORTANT

To guarantee that a critical change on the primary database is replicated to a secondary before you failover, you can force synchronization to ensure the replication of critical changes (for example, password updates). Forced synchronization impacts performance because it blocks the calling thread until all committed transactions are replicated. For details, see [sp_wait_for_database_copy_sync](#). To monitor the replication lag between the primary database and geo-secondary, see [sys.dm_geo_replication_link_status](#).

The following figure shows an example of active geo-replication configured with a primary in the North Central US region and secondary in the South Central US region.

PRIMARY		
	North Central US	auditing-server/dimadhus_d111
Online		

SECONDARIES		
	South Central US	sqlloctest/dimadhus_d111
Readable ...		

TARGET REGIONS		
	West US	
	Central US	
	South Central US	
	North Central US	
	East US	
	East US 2	
	Brazil South	
	North Europe	

Because the secondary databases are readable, they can be used to offload read-only workloads such as reporting jobs. If you are using active geo-replication, it is possible to create the secondary database in the same region with the primary, but it does not increase the application's resilience to catastrophic failures. If you are using auto-failover groups, your secondary database is always created in a different region.

In addition to disaster recovery active geo-replication can be used in the following scenarios:

- **Database migration:** You can use active geo-replication to migrate a database from one server to another online with minimum downtime.
- **Application upgrades:** You can create an extra secondary as a fail back copy during application upgrades.

To achieve real business continuity, adding database redundancy between datacenters is only part of the solution. Recovering an application (service) end-to-end after a catastrophic failure requires recovery of all components that constitute the service and any dependent services. Examples of these components include the client software (for example, a browser with a custom JavaScript), web front ends, storage, and DNS. It is critical that all components are resilient to the same failures and become available within the recovery time objective (RTO) of your application. Therefore, you need to identify all dependent services and understand the guarantees and capabilities they provide. Then, you must take adequate steps to ensure that your service functions during the failover of the services on which it depends. For more information about designing solutions for disaster recovery, see [Designing Cloud Solutions for Disaster Recovery Using active geo-replication](#).

Active geo-replication terminology and capabilities

- **Automatic Asynchronous Replication**

You can only create a secondary database by adding to an existing database. The secondary can be created in any Azure SQL Database server. Once created, the secondary database is populated with the data copied from the primary database. This process is known as seeding. After secondary database has been created and seeded, updates to the primary database are asynchronously replicated to the secondary database automatically. Asynchronous replication means that transactions are committed on the primary database before they are replicated to the secondary database.

- **Readable secondary databases**

An application can access a secondary database for read-only operations using the same or different security principals used for accessing the primary database. The secondary databases operate in snapshot isolation mode to ensure replication of the updates of the primary (log replay) is not delayed by queries executed on the secondary.

NOTE

The log replay is delayed on the secondary database if there are schema updates on the Primary. The latter requires a schema lock on the secondary database.

IMPORTANT

You can use geo-replication to create a secondary database in the same region as the primary. You can use this secondary to load-balance a read-only workloads in the same region. However, a secondary database in the same region does not provide additional fault resilience and therefore is not a suitable failover target for disaster recovery. It will also not guarantee availability zone isolation. Use Business critical or Premium service tier with [zone redundant configuration](#) to achieve availability zone isolation.

- **Planned failover**

Planned failover switches the roles of primary and secondary databases after the full synchronization is completed. It is an online operation that does not result in data loss. The time of the operation depends on the size of the transaction log on the primary that needs to be synchronized. Planned failover is designed for following scenarios: (a) to perform DR drills in production when the data loss is not acceptable; (b) to relocate the database to a different region; and (c) to return the database to the primary region after the outage has been mitigated (failback).

- **Unplanned failover**

Unplanned or forced failover immediately switches the secondary to the primary role without any synchronization with the primary. Any transactions committed to the primary but not replicated to the secondary will be lost. This operation is designed as a recovery method during outages when the primary is not accessible, but the database availability must be quickly restored. When the original primary is back online it will automatically re-connect and become a new secondary. All unsynchronized transactions before the failover will be preserved in the backup file but will not be synchronized with the new primary to avoid conflicts. These transactions will have to be manually merged with the most recent version of the primary database.

- **Multiple readable secondaries**

Up to 4 secondary databases can be created for each primary. If there is only one secondary database, and it fails, the application is exposed to higher risk until a new secondary database is created. If multiple secondary databases exist, the application remains protected even if one of the secondary databases fails. The additional secondaries can also be used to scale out the read-only workloads

NOTE

If you are using active geo-replication to build a globally distributed application and need to provide read-only access to data in more than four regions, you can create secondary of a secondary (a process known as chaining). This way you can achieve virtually unlimited scale of database replication. In addition, chaining reduces the overhead of replication from the primary database. The trade-off is the increased replication lag on the leaf-most secondary databases.

- **Geo-replication of databases in an elastic pool**

Each secondary database can separately participate in an elastic pool or not be in any elastic pool at all. The pool choice for each secondary database is separate and does not depend upon the configuration of any other secondary database (whether primary or secondary). Each elastic pool is contained within a single region, therefore multiple secondary databases in the same topology can never share an elastic pool.

- **User-controlled failover and fallback**

A secondary database can explicitly be switched to the primary role at any time by the application or the user. During a real outage the “unplanned” option should be used, which immediately promotes a secondary to be the primary. When the failed primary recovers and is available again, the system automatically marks the recovered primary as a secondary and bring it up-to-date with the new primary. Due to the asynchronous nature of replication, a small amount of data can be lost during unplanned failovers if a primary fails before it replicates the most recent changes to the secondary. When a primary with multiple secondaries fails over, the system automatically reconfigures the replication relationships and links the remaining secondaries to the newly promoted primary without requiring any user intervention. After the outage that caused the failover is mitigated, it may be desirable to return the application to the primary region. To do that, the failover command should be invoked with the “planned” option.

Preparing secondary database for failover

To ensure that your application can immediately access the new primary after failover, ensure the authentication requirements for your secondary server and database are properly configured. For details, see [SQL Database security after disaster recovery](#). To guarantee compliance after failover, make sure that the backup retention policy on the secondary database matches that of the primary. These settings are not part of the database and are not replicated. By default, the secondary will be configured with a default PITR retention period of seven days. For details, see [SQL Database automated backups](#).

IMPORTANT

If your database is a member of a failover group, you cannot initiate its failover using the geo-replication failover command. Consider using failover command for the group. If you need to failover an individual database, you must remove it from the failover group first. See [failover groups](#) for details.

Configuring secondary database

Both primary and secondary databases are required to have the same service tier. It is also strongly recommended that secondary database is created with the same compute size (DTUs or vCores) as the primary. If the primary database is experiencing a heavy write workload, a secondary with lower compute size may not be able to keep up with it. It will cause the redo lag on the secondary and potential unavailability. A secondary database that is lagging behind the primary also risks a large data loss should a forced failover be required. To mitigate these risks, effective active geo-replication will throttle the primary's log rate to allow its

secondaries to catch up. The other consequence of an imbalanced secondary configuration is that after failover the application's performance will suffer due to insufficient compute capacity of the new primary. It will be required to upgrade to a higher compute to the necessary level, which will not be possible until the outage is mitigated.

IMPORTANT

The published RPO = 5 sec cannot be guaranteed unless the secondary database is configured with the same compute size as the primary.

If you decide to create the secondary with lower compute size, the log IO percentage chart on Azure portal provides a good way to estimate the minimal compute size of the secondary that is required to sustain the replication load. For example, if your Primary database is P6 (1000 DTU) and its log IO percent is 50% the secondary needs to be at least P4 (500 DTU). You can also retrieve the log IO data using [sys.resource_stats](#) or [sys.dm_db_resource_stats](#) database views. The throttling is reported as a HADR_THROTTLE_LOG_RATE_MISMATCHED_SLO wait state in the [sys.dm_exec_requests](#) and [sys.dm_os_wait_stats](#) database views.

For more information on the SQL Database compute sizes, see [What are SQL Database Service Tiers](#).

Cross-subscription geo-replication

To setup active geo-replication between two databases belonging to different subscriptions (whether under the same tenant or not), you must follow the special procedure described in this section. The procedure is based on SQL commands and requires:

- Creating a privileged login on both servers
- Adding the IP address to the allow list of the client performing the change on both servers (such as the IP address of the host running SQL Server Management Studio).

The client performing the changes needs network access to the primary server. Although the same IP address of the client must be added to the allow list on the secondary server, network connectivity to the secondary server is not strictly required.

On the master of the primary server

1. Add the IP address to the allow list of the client performing the changes (for more information see, [Configure firewall](#)).
2. Create a login dedicated to setup active geo-replication (and adjust the credentials as needed):

```
create login geodrsetup with password = 'ComplexPassword01'
```

3. Create a corresponding user and assign it to the dbmanager role:

```
create user geodrsetup for login geodrsetup
alter role geodrsetup dbmanager add member geodrsetup
```

4. Take note of the SID of the new login using this query:

```
select sid from sys.sql_logins where name = 'geodrsetup'
```

On the source database on the primary server

1. Create a user for the same login:

```
create user geodrsetup for login geodrsetup
```

2. Add the user to the db_owner role:

```
alter role db_owner add member geodrsetup
```

On the master of the secondary server

1. Add the IP address to the allow list of the client performing the changes. It must be the same exact IP address of the primary server.
2. Create the same login as on the primary server, using the same username password, and SID:

```
create login geodrsetup with password = 'ComplexPassword01',
sid=0x01060000000000640000000000000001C98F52B95D9C84BBBA8578FACE37C3E
```

3. Create a corresponding user and assign it to the dbmanager role:

```
create user geodrsetup for login geodrsetup;
alter role dbmanager add member geodrsetup
```

On the master of the primary server

1. Login to the master of the primary server using the new login.
2. Create a secondary replica of the source database on the secondary server (adjust database name and servername as needed):

```
alter database dbrep add secondary on server <servername>
```

After the initial setup, the users, logins, and firewall rules created can be removed.

Keeping credentials and firewall rules in sync

We recommend using [database-level IP firewall rules](#) for geo-replicated databases so these rules can be replicated with the database to ensure all secondary databases have the same IP firewall rules as the primary. This approach eliminates the need for customers to manually configure and maintain firewall rules on servers hosting both the primary and secondary databases. Similarly, using [contained database users](#) for data access ensures both primary and secondary databases always have the same user credentials so during a failover, there is no disruptions due to mismatches with logins and passwords. With the addition of [Azure Active Directory](#), customers can manage user access to both primary and secondary databases and eliminating the need for managing credentials in databases altogether.

Upgrading or downgrading primary database

You can upgrade or downgrade a primary database to a different compute size (within the same service tier, not between General Purpose and Business Critical) without disconnecting any secondary databases. When upgrading, we recommend that you upgrade the secondary database first, and then upgrade the primary. When downgrading, reverse the order: downgrade the primary first, and then downgrade the secondary. When you upgrade or downgrade the database to a different service tier, this recommendation is enforced.

NOTE

If you created secondary database as part of the failover group configuration it is not recommended to downgrade the secondary database. This is to ensure your data tier has sufficient capacity to process your regular workload after failover is activated.

IMPORTANT

The primary database in a failover group can't scale to a higher tier unless the secondary database is first scaled to the higher tier. If you try to scale the primary database before the secondary database is scaled, you might receive the following error:

```
Error message: The source database 'Primaryserver.DBName' cannot have higher edition than the target database 'Secondaryserver.DBName'. Upgrade the edition on the target before upgrading the source.
```

Preventing the loss of critical data

Due to the high latency of wide area networks, continuous copy uses an asynchronous replication mechanism. Asynchronous replication makes some data loss unavoidable if a failure occurs. However, some applications may require no data loss. To protect these critical updates, an application developer can call the [sp_wait_for_database_copy_sync](#) system procedure immediately after committing the transaction. Calling **sp_wait_for_database_copy_sync** blocks the calling thread until the last committed transaction has been transmitted to the secondary database. However, it does not wait for the transmitted transactions to be replayed and committed on the secondary. **sp_wait_for_database_copy_sync** is scoped to a specific continuous copy link. Any user with the connection rights to the primary database can call this procedure.

NOTE

sp_wait_for_database_copy_sync prevents data loss after failover, but does not guarantee full synchronization for read access. The delay caused by a **sp_wait_for_database_copy_sync** procedure call can be significant and depends on the size of the transaction log at the time of the call.

Monitoring geo-replication lag

To monitor lag with respect to RPO, use *replication_lag_sec* column of [sys.dm_geo_replication_link_status](#) on the primary database. It shows lag in seconds between the transactions committed on the primary and persisted on the secondary. E.g. if the value of the lag is 1 second, it means if the primary is impacted by an outage at this moment and failover is initiated, 1 second of the most recent transitions will not be saved.

To measure lag with respect to changes on the primary database that have been applied on the secondary, i.e. available to read from the secondary, compare *last_commit* time on the secondary database with the same value on the primary database.

NOTE

Sometimes *replication_lag_sec* on the primary database has a NULL value, which means that the primary does not currently know how far the secondary is. This typically happens after process restarts and should be a transient condition. Consider alerting the application if the *replication_lag_sec* returns NULL for an extended period of time. It would indicate that the secondary database cannot communicate with the primary due to a permanent connectivity failure. There are also conditions that could cause the difference between *last_commit* time on the secondary and on the primary database to become large. E.g. if a commit is made on the primary after a long period of no changes, the difference will jump up to a large value before quickly returning to 0. Consider it an error condition when the difference between these two values remains large for a long time.

Programmatically managing active geo-replication

As discussed previously, active geo-replication can also be managed programmatically using Azure PowerShell and the REST API. The following tables describe the set of commands available. Active geo-replication includes a set of Azure Resource Manager APIs for management, including the [Azure SQL Database REST API](#) and [Azure PowerShell cmdlets](#). These APIs require the use of resource groups and support role-based security (RBAC). For more information on how to implement access roles, see [Azure Role-Based Access Control](#).

T-SQL: Manage failover of single and pooled databases

IMPORTANT

These Transact-SQL commands only apply to active geo-replication and do not apply to failover groups. As such, they also do not apply to Managed Instances, as they only support failover groups.

COMMAND	DESCRIPTION
ALTER DATABASE	Use ADD SECONDARY ON SERVER argument to create a secondary database for an existing database and starts data replication
ALTER DATABASE	Use FAILOVER or FORCE_FAILOVER_ALLOW_DATA_LOSS to switch a secondary database to be primary to initiate failover
ALTER DATABASE	Use REMOVE SECONDARY ON SERVER to terminate a data replication between a SQL Database and the specified secondary database.
sys.geo_replication_links	Returns information about all existing replication links for each database on the Azure SQL Database server.
sys.dm_geo_replication_link_status	Gets the last replication time, last replication lag, and other information about the replication link for a given SQL database.
sys.dm_operation_status	Shows the status for all database operations including the status of the replication links.
sp_wait_for_database_copy_sync	causes the application to wait until all committed transactions are replicated and acknowledged by the active secondary database.

COMMAND	DESCRIPTION

PowerShell: Manage failover of single and pooled databases

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

CMDLET	DESCRIPTION
Get-AzSqlDatabase	Gets one or more databases.
New-AzSqlDatabaseSecondary	Creates a secondary database for an existing database and starts data replication.
Set-AzSqlDatabaseSecondary	Switches a secondary database to be primary to initiate failover.
Remove-AzSqlDatabaseSecondary	Terminates data replication between a SQL Database and the specified secondary database.
Get-AzSqlDatabaseReplicationLink	Gets the geo-replication links between an Azure SQL Database and a resource group or SQL Server.

IMPORTANT

For sample scripts, see [Configure and failover a single database using active geo-replication](#) and [Configure and failover a pooled database using active geo-replication](#).

REST API: Manage failover of single and pooled databases

API	DESCRIPTION
Create or Update Database (createMode=Restore)	Creates, updates, or restores a primary or a secondary database.
Get Create or Update Database Status	Returns the status during a create operation.
Set Secondary Database as Primary (Planned Failover)	Sets which secondary database is primary by failing over from the current primary database. This option is not supported for Managed Instance.

API	DESCRIPTION
Set Secondary Database as Primary (Unplanned Failover)	Sets which secondary database is primary by failing over from the current primary database. This operation might result in data loss. This option is not supported for Managed Instance.
Get Replication Link	Gets a specific replication link for a given SQL database in a geo-replication partnership. It retrieves the information visible in the sys.geo_replication_links catalog view. This option is not supported for Managed Instance.
Replication Links - List By Database	Gets all replication links for a given SQL database in a geo-replication partnership. It retrieves the information visible in the sys.geo_replication_links catalog view.
Delete Replication Link	Deletes a database replication link. Cannot be done during failover.

Next steps

- For sample scripts, see:
 - [Configure and failover a single database using active geo-replication](#)
 - [Configure and failover a pooled database using active geo-replication](#)
- SQL Database also supports auto-failover groups. For more information, see [using auto-failover groups](#).
- For a business continuity overview and scenarios, see [Business continuity overview](#)
- To learn about Azure SQL Database automated backups, see [SQL Database automated backups](#).
- To learn about using automated backups for recovery, see [Restore a database from the service-initiated backups](#).
- To learn about authentication requirements for a new primary server and database, see [SQL Database security after disaster recovery](#).

Use auto-failover groups to enable transparent and coordinated failover of multiple databases

2/10/2020 • 35 minutes to read • [Edit Online](#)

Auto-failover groups is a SQL Database feature that allows you to manage replication and failover of a group of databases on a SQL Database server or all databases in a managed instance to another region. It is a declarative abstraction on top of the existing [active geo-replication](#) feature, designed to simplify deployment and management of geo-replicated databases at scale. You can initiate failover manually or you can delegate it to the SQL Database service based on a user-defined policy. The latter option allows you to automatically recover multiple related databases in a secondary region after a catastrophic failure or other unplanned event that results in full or partial loss of the SQL Database service's availability in the primary region. A failover group can include one or multiple databases, typically used by the same application. Additionally, you can use the readable secondary databases to offload read-only query workloads. Because auto-failover groups involve multiple databases, these databases must be configured on the primary server. Auto-failover groups support replication of all databases in the group to only one secondary server in a different region.

NOTE

When working with single or pooled databases on a SQL Database server and you want multiple secondaries in the same or different regions, use [active geo-replication](#).

When you are using auto-failover groups with automatic failover policy, any outage that impacts one or several of the databases in the group results in automatic failover. Typically these are incidents that cannot be self-mitigated by the built-in automatic high availability operations. The examples of failover triggers include an incident caused by a SQL tenant ring or control ring being down due to an OS kernel memory leak on several compute nodes, or an incident caused by one or more tenant rings being down because a wrong network cable was cut during routine hardware decommissioning. For more information, see [SQL Database High Availability](#).

In addition, auto-failover groups provide read-write and read-only listener end-points that remain unchanged during failovers. Whether you use manual or automatic failover activation, failover switches all secondary databases in the group to primary. After the database failover is completed, the DNS record is automatically updated to redirect the endpoints to the new region. For the specific RPO and RTO data, see [Overview of Business Continuity](#).

When you are using auto-failover groups with automatic failover policy, any outage that impacts databases in the SQL Database server or managed instance results in automatic failover. You can manage auto-failover group using:

- [Azure portal](#)
- [Azure CLI: Failover Group](#)
- [PowerShell: Failover Group](#)
- [REST API: Failover group](#).

After failover, ensure the authentication requirements for your server and database are configured on the new primary. For details, see [SQL Database security after disaster recovery](#).

To achieve real business continuity, adding database redundancy between datacenters is only part of the

solution. Recovering an application (service) end-to-end after a catastrophic failure requires recovery of all components that constitute the service and any dependent services. Examples of these components include the client software (for example, a browser with a custom JavaScript), web front ends, storage, and DNS. It is critical that all components are resilient to the same failures and become available within the recovery time objective (RTO) of your application. Therefore, you need to identify all dependent services and understand the guarantees and capabilities they provide. Then, you must take adequate steps to ensure that your service functions during the failover of the services on which it depends. For more information about designing solutions for disaster recovery, see [Designing Cloud Solutions for Disaster Recovery](#) [Using active geo-replication](#).

Auto-failover group terminology and capabilities

- **Failover group (FOG)**

A failover group is a named group of databases managed by a single SQL Database server or within a single managed instance that can fail over as a unit to another region in case all or some primary databases become unavailable due to an outage in the primary region. When created for managed instances, a failover group contains all user databases in the instance and therefore only one failover group can be configured on an instance.

IMPORTANT

The name of the failover group must be globally unique within the `.database.windows.net` domain.

- **SQL Database servers**

With SQL Database servers, some or all of the user databases on a single SQL Database server can be placed in a failover group. Also, a SQL Database server supports multiple failover groups on a single SQL Database server.

- **Primary**

The SQL Database server or managed instance that hosts the primary databases in the failover group.

- **Secondary**

The SQL Database server or managed instance that hosts the secondary databases in the failover group. The secondary cannot be in the same region as the primary.

- **Adding single databases to failover group**

You can put several single databases on the same SQL Database server into the same failover group. If you add a single database to the failover group, it automatically creates a secondary database using the same edition and compute size on secondary server. You specified that server when the failover group was created. If you add a database that already has a secondary database in the secondary server, that geo-replication link is inherited by the group. When you add a database that already has a secondary database in a server that is not part of the failover group, a new secondary is created in the secondary server.

IMPORTANT

Make sure that the secondary server doesn't have a database with the same name unless it is an existing secondary database. In failover groups for managed instance all user databases are replicated. You cannot pick a subset of user databases for replication in the failover group.

- **Adding databases in elastic pool to failover group**

You can put all or several databases within an elastic pool into the same failover group. If the primary database is in an elastic pool, the secondary is automatically created in the elastic pool with the same name (secondary pool). You must ensure that the secondary server contains an elastic pool with the same exact name and enough free capacity to host the secondary databases that will be created by the failover group. If you add a database in the pool that already has a secondary database in the secondary pool, that geo-replication link is inherited by the group. When you add a database that already has a secondary database in a server that is not part of the failover group, a new secondary is created in the secondary pool.

- **Initial Seeding**

When adding databases, elastic pools, or managed instances to a failover group, there is an initial seeding phase before data replication starts. The initial seeding phase is the longest and most expensive operation. Once initial seeding completes, data is synchronized, and then only subsequent data changes are replicated. The time it takes for the initial seed to complete depends on the size of your data, number of replicated databases, and the speed of the link between the entities in the failover group. Under normal circumstances, typical seeding speed is 50-500 GB an hour for a single database or elastic pool, and 18-35 GB an hour for a managed instance. Seeding is performed for all databases in parallel. You can use the stated seeding speed, along with the number of databases and the total size of data to estimate how long the initial seeding phase will take before data replication starts.

For managed instances, the speed of the Express Route link between the two instances also needs to be considered when estimating the time of the initial seeding phase. If the speed of the link between the two instances is slower than what is necessary, the time to seed is likely to be notably impacted. You can use the stated seeding speed, number of databases, total size of data, and the link speed to estimate how long the initial seeding phase will take before data replication starts. For example, for a single 100 GB database, the initial seed phase would take anywhere from 2.8 - 5.5 hours if the link is capable of pushing 35 GB per hour. If the link can only transfer 10 GB per hour, then seeding a 100 GB database will take about 10 hours. If there are multiple databases to replicate, seeding will be executed in parallel, and, when combined with a slow link speed, the initial seeding phase may take considerably longer, especially if the parallel seeding of data from all databases exceeds the available link bandwidth. If the network bandwidth between two instances is limited and you are adding multiple managed instances to a failover group, consider adding multiple managed instances to the failover group sequentially, one by one.

- **DNS zone**

A unique ID that is automatically generated when a new instance is created. A multi-domain (SAN) certificate for this instance is provisioned to authenticate the client connections to any instance in the same DNS zone. The two managed instances in the same failover group must share the DNS zone.

NOTE

A DNS zone ID is not required for failover groups created for SQL Database servers.

- **Failover group read-write listener**

A DNS CNAME record that points to the current primary's URL. It is created automatically when the failover group is created and allows the read-write SQL workload to transparently reconnect to the primary database when the primary changes after failover. When the failover group is created on a SQL Database server, the DNS CNAME record for the listener URL is formed as

`<fog-name>.database.windows.net`. When the failover group is created on a managed instance, the DNS CNAME record for the listener URL is formed as `<fog-name>.zone_id.database.windows.net`.

- **Failover group read-only listener**

A DNS CNAME record formed that points to the read-only listener that points to the secondary's URL. It is created automatically when the failover group is created and allows the read-only SQL workload to transparently connect to the secondary using the specified load-balancing rules. When the failover group is created on a SQL Database server, the DNS CNAME record for the listener URL is formed as `<fog-name>.secondary.database.windows.net`. When the failover group is created on a managed instance, the DNS CNAME record for the listener URL is formed as `<fog-name>.zone_id.secondary.database.windows.net`.

- **Automatic failover policy**

By default, a failover group is configured with an automatic failover policy. The SQL Database service triggers failover after the failure is detected and the grace period has expired. The system must verify that the outage cannot be mitigated by the built-in [high availability infrastructure of the SQL Database service](#) due to the scale of the impact. If you want to control the failover workflow from the application, you can turn off automatic failover.

NOTE

Because verification of the scale of the outage and how quickly it can be mitigated involves human actions by the operations team, the grace period cannot be set below one hour. This limitation applies to all databases in the failover group regardless of their data synchronization state.

- **Read-only failover policy**

By default, the failover of the read-only listener is disabled. It ensures that the performance of the primary is not impacted when the secondary is offline. However, it also means the read-only sessions will not be able to connect until the secondary is recovered. If you cannot tolerate downtime for the read-only sessions and are OK to temporarily use the primary for both read-only and read-write traffic at the expense of the potential performance degradation of the primary, you can enable failover for the read-only listener by configuring the `AllowReadOnlyFailoverToPrimary` property. In that case, the read-only traffic will be automatically redirected to the primary if the secondary is not available.

- **Planned failover**

Planned failover performs full synchronization between primary and secondary databases before the secondary switches to the primary role. This guarantees no data loss. Planned failover is used in the following scenarios:

- Perform disaster recovery (DR) drills in production when the data loss is not acceptable
- Relocate the databases to a different region
- Return the databases to the primary region after the outage has been mitigated (failback).

- **Unplanned failover**

Unplanned or forced failover immediately switches the secondary to the primary role without any synchronization with the primary. This operation will result in data loss. Unplanned failover is used as a recovery method during outages when the primary is not accessible. When the original primary is back online, it will automatically reconnect without synchronization and become a new secondary.

- **Manual failover**

You can initiate failover manually at any time regardless of the automatic failover configuration. If automatic failover policy is not configured, manual failover is required to recover databases in the failover group to the secondary. You can initiate forced or friendly failover (with full data synchronization). The latter could be used to relocate the primary to the secondary region. When failover is completed, the DNS records are automatically updated to ensure connectivity to the new primary.

- **Grace period with data loss**

Because the primary and secondary databases are synchronized using asynchronous replication, the failover may result in data loss. You can customize the automatic failover policy to reflect your application's tolerance to data loss. By configuring `GracePeriodWithDataLossHours`, you can control how long the system waits before initiating the failover that is likely to result data loss.

- **Multiple failover groups**

You can configure multiple failover groups for the same pair of servers to control the scale of failovers. Each group fails over independently. If your multi-tenant application uses elastic pools, you can use this capability to mix primary and secondary databases in each pool. This way you can reduce the impact of an outage to only half of the tenants.

NOTE

Managed Instance does not support multiple failover groups.

Permissions

Permissions for a failover group are managed via [role-based access control \(RBAC\)](#). The [SQL Server Contributor](#) role has all the necessary permissions to manage failover groups.

Create failover group

To create a failover group, you need RBAC write access to both the primary and secondary servers, and to all databases in the failover group. For a managed instance, you need RBAC write access to both the primary and secondary managed instance, but permissions on individual databases are not relevant since individual managed instance databases cannot be added to or removed from a failover group.

Update a failover group

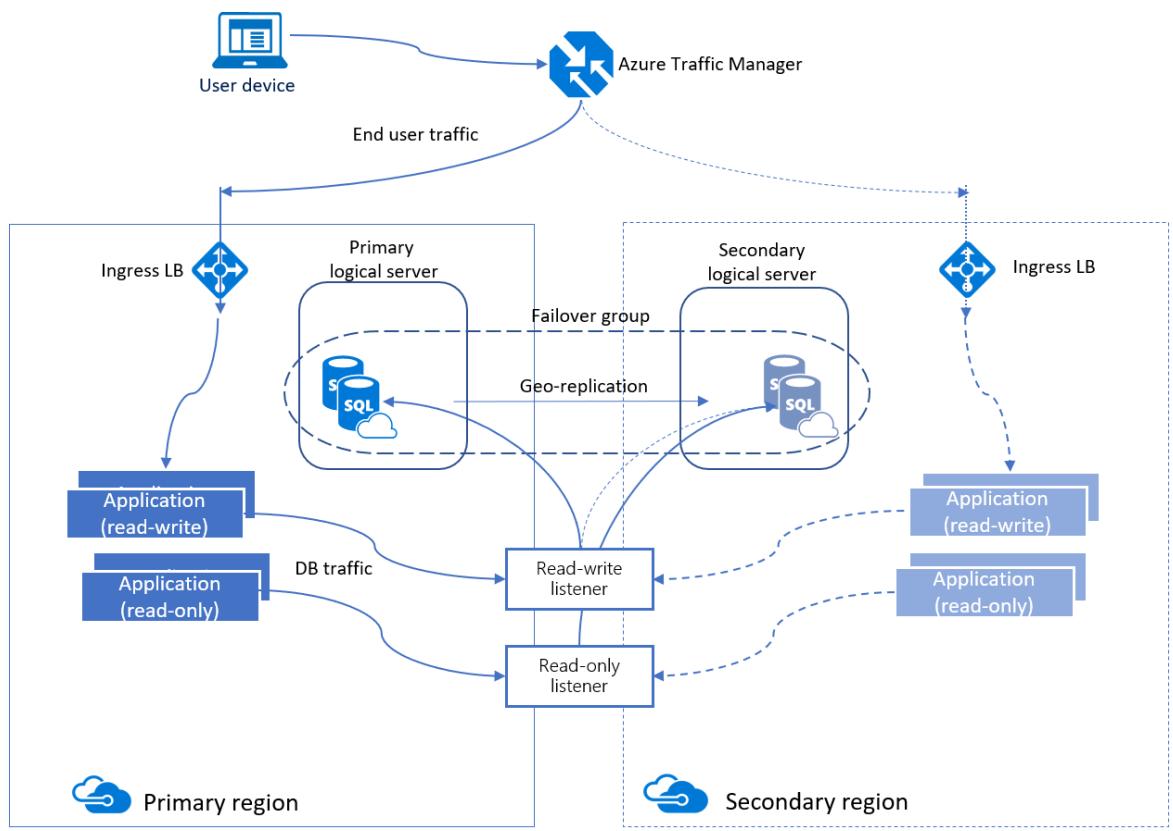
To update a failover group, you need RBAC write access to the failover group, and all databases on the current primary server or managed instance.

Failover a failover group

To fail over a failover group, you need RBAC write access to the failover group on the new primary server or managed instance.

Best practices of using failover groups with single databases and elastic pools

The auto-failover group must be configured on the primary SQL Database server and will connect it to the secondary SQL Database server in a different Azure region. The groups can include all or some databases in these servers. The following diagram illustrates a typical configuration of a geo-redundant cloud application using multiple databases and auto-failover group.



NOTE

See [Add single database to a failover group](#) for a detailed step-by-step tutorial adding a single database to a failover group.

When designing a service with business continuity in mind, follow these general guidelines:

Using one or several failover groups to manage failover of multiple databases

One or many failover groups can be created between two servers in different regions (primary and secondary servers). Each group can include one or several databases that are recovered as a unit in case all or some primary databases become unavailable due to an outage in the primary region. The failover group creates geo-secondary database with the same service objective as the primary. If you add an existing geo-replication relationship to the failover group, make sure the geo-secondary is configured with the same service tier and compute size as the primary.

IMPORTANT

Creating failover groups between two servers in different subscriptions is not currently supported for single databases and elastic pools. If you move the primary or secondary server to a different subscription after the failover group has been created, it could result in failures of the failover requests and other operations.

Using read-write listener for OLTP workload

When performing OLTP operations, use `<fog-name>.database.windows.net` as the server URL and the connections are automatically directed to the primary. This URL does not change after the failover. Note the failover involves updating the DNS record so the client connections are redirected to the new primary only after the client DNS cache is refreshed.

Using read-only listener for read-only workload

If you have a logically isolated read-only workload that is tolerant to certain staleness of data, you can use the secondary database in the application. For read-only sessions, use

`<fog-name>.secondary.database.windows.net` as the server URL and the connection is automatically directed to the secondary. It is also recommended that you indicate in connection string read intent by using `ApplicationIntent=ReadOnly`. If you want to ensure that the read-only workload can reconnect after failover or in case the secondary server goes offline, make sure to configure the `AllowReadOnlyFailoverToPrimary` property of the failover policy.

Preparing for performance degradation

A typical Azure application uses multiple Azure service and consists of multiple components. The automated failover of the failover group is triggered based on the state the Azure SQL components alone. Other Azure services in the primary region may not be affected by the outage and their components may still be available in that region. Once the primary databases switch to the DR region, the latency between the dependent components may increase. To avoid the impact of higher latency on the application's performance, ensure the redundancy of all the application's components in the DR region and follow these [network security guidelines](#).

Preparing for data loss

If an outage is detected, SQL waits for the period you specified by `GracePeriodWithDataLossHours`. The default value is 1 hour. If you cannot afford data loss, make sure to set `GracePeriodWithDataLossHours` to a sufficiently large number, such as 24 hours. Use manual group failover to fail back from the secondary to the primary.

IMPORTANT

Elastic pools with 800 or fewer DTUs and more than 250 databases using geo-replication may encounter issues including longer planned failovers and degraded performance. These issues are more likely to occur for write intensive workloads, when geo-replication endpoints are widely separated by geography, or when multiple secondary endpoints are used for each database. Symptoms of these issues are indicated when the geo-replication lag increases over time. This lag can be monitored using `sys.dm_geo_replication_link_status`. If these issues occur, then mitigations include increasing the number of pool DTUs, or reducing the number of geo-replicated databases in the same pool.

Changing secondary region of the failover group

To illustrate the change sequence, we will assume that server A is the primary server, server B is the existing secondary server, and server C is the new secondary in the third region. To make the transition, follow these steps:

1. Create additional secondaries of each database on server A to server C using [active geo-replication](#). Each database on server A will have two secondaries, one on server B and one on server C. This will guarantee that the primary databases remain protected during the transition.
2. Delete the failover group. At this point the logins will be failing. This is because the SQL aliases for the failover group listeners have been deleted and the gateway will not recognize the failover group name.
3. Re-create the failover group with the same name between servers A and C. At this point the logins will stop failing.
4. Add all primary databases on server A to the new failover group.
5. Drop server B. All databases on B will be deleted automatically.

Changing primary region of the failover group

To illustrate the change sequence, we will assume server A is the primary server, server B is the existing secondary server, and server C is the new primary in the third region. To make the transition, follow these steps:

1. Perform a planned failover to switch the primary server to B. Server A will become the new secondary server. The failover may result in several minutes of downtime. The actual time will depend on the size

of failover group.

2. Create additional secondaries of each database on server B to server C using [active geo-replication](#). Each database on server B will have two secondaries, one on server A and one on server C. This will guarantee that the primary databases remain protected during the transition.
3. Delete the failover group. At this point the logins will be failing. This is because the SQL aliases for the failover group listeners have been deleted and the gateway will not recognize the failover group name.
4. Re-create the failover group with the same name between servers A and C. At this point the logins will stop failing.
5. Add all primary databases on B to the new failover group.
6. Perform a planned failover of the failover group to switch B and C. Now server C will become the primary and B - the secondary. All secondary databases on server A will be automatically linked to the primaries on C. As in step 1, the failover may result in several minutes of downtime.
7. Drop the server A. All databases on A will be deleted automatically.

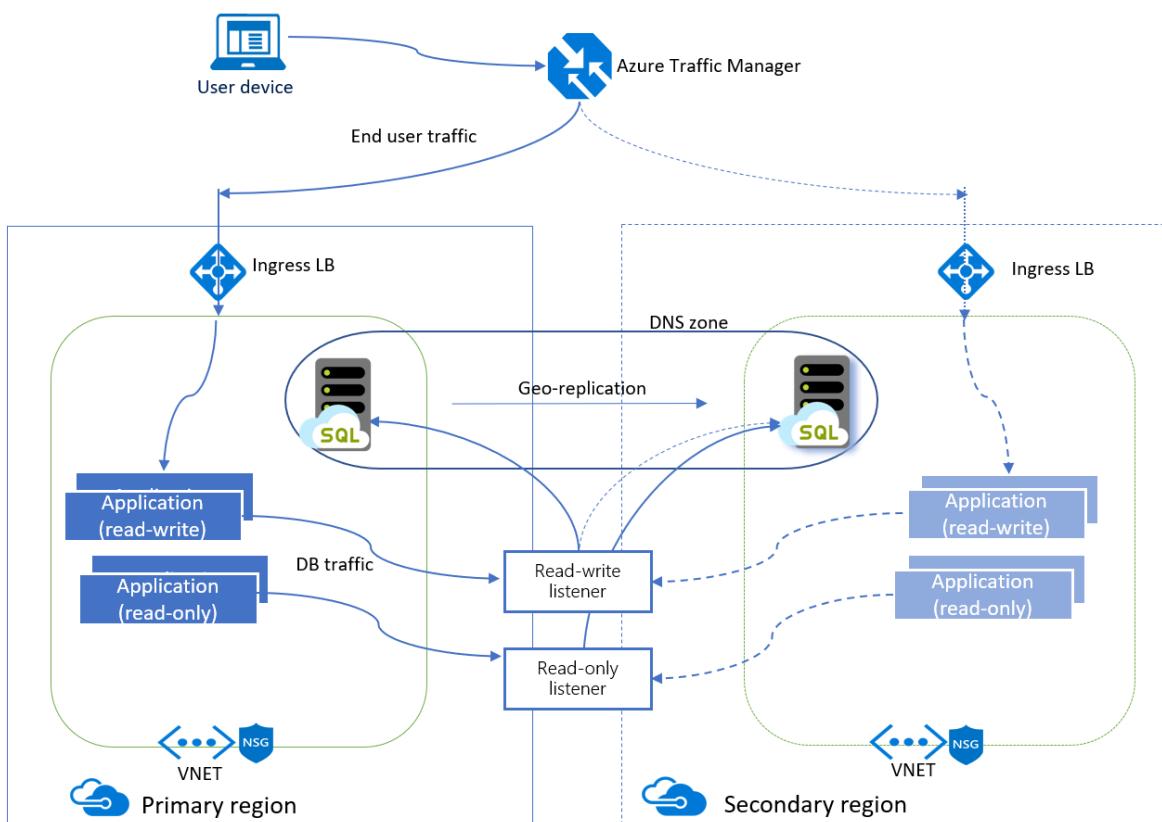
IMPORTANT

When the failover group is deleted, the DNS records for the listener endpoints are also deleted. At that point, there is a non-zero probability of somebody else creating a failover group or server alias with the same name, which will prevent you from using it again. To minimize the risk, don't use generic failover group names.

Best practices of using failover groups with managed instances

The auto-failover group must be configured on the primary instance and will connect it to the secondary instance in a different Azure region. All databases in the instance will be replicated to the secondary instance.

The following diagram illustrates a typical configuration of a geo-redundant cloud application using managed instance and auto-failover group.



NOTE

See [Add managed instance to a failover group](#) for a detailed step-by-step tutorial adding a managed instance to use failover group.

If your application uses managed instance as the data tier, follow these general guidelines when designing for business continuity:

Creating the secondary instance

To ensure non-interrupted connectivity to the primary instance after failover both the primary and secondary instances must be in the same DNS zone. It will guarantee that the same multi-domain (SAN) certificate can be used to authenticate the client connections to either of the two instances in the failover group. When your application is ready for production deployment, create a secondary instance in a different region and make sure it shares the DNS zone with the primary instance. You can do it by specifying a `DNS Zone Partner` optional parameter using the Azure portal, PowerShell, or the REST API.

IMPORTANT

First instance created in the subnet determines DNS zone for all subsequent instances in the same subnet. This means that two instances from the same subnet cannot belong to different DNS zones.

For more information about creating the secondary instance in the same DNS zone as the primary instance, see [Create a secondary managed instance](#).

Enabling replication traffic between two instances

Because each instance is isolated in its own VNet, two-directional traffic between these VNets must be allowed. See [Azure VPN gateway](#)

Creating a failover group between managed instances in different subscriptions

You can create a failover group between managed instances in two different subscriptions. When using PowerShell API you can do it by specifying the `PartnerSubscriptionId` parameter for the secondary instance. When using REST API, each instance ID included in the `properties.managedInstancePairs` parameter can have its own subscriptionID.

IMPORTANT

Azure portal does not support creation of failover groups across different subscriptions. Also, for the existing failover groups across different subscriptions and/or resource groups, failover cannot be initiated manually via portal from the primary instance. Initiate it from the geo-secondary instance instead.

Managing failover to secondary instance

The failover group will manage the failover of all the databases in the instance. When a group is created, each database in the instance will be automatically geo-replicated to the secondary instance. You cannot use failover groups to initiate a partial failover of a subset of the databases.

IMPORTANT

If a database is removed from the primary instance, it will also be dropped automatically on the geo secondary instance.

Using read-write listener for OLTP workload

When performing OLTP operations, use `<fog-name>.zone_id.database.windows.net` as the server URL and the connections are automatically directed to the primary. This URL does not change after the failover. The failover involves updating the DNS record, so the client connections are redirected to the new primary only after the client DNS cache is refreshed. Because the secondary instance shares the DNS zone with the primary, the client application will be able to reconnect to it using the same SAN certificate.

Using read-only listener to connect to the secondary instance

If you have a logically isolated read-only workload that is tolerant to certain staleness of data, you can use the secondary database in the application. To connect directly to the geo-replicated secondary, use `server.secondary.zone_id.database.windows.net` as the server URL and the connection is made directly to the geo-replicated secondary.

NOTE

In certain service tiers, Azure SQL Database supports the use of [read-only replicas](#) to load balance read-only query workloads using the capacity of one read-only replica and using the `ApplicationIntent=ReadOnly` parameter in the connection string. When you have configured a geo-replicated secondary, you can use this capability to connect to either a read-only replica in the primary location or in the geo-replicated location.

- To connect to a read-only replica in the primary location, use `<fog-name>.zone_id.database.windows.net`.
- To connect to a read-only replica in the secondary location, use `<fog-name>.secondary.zone_id.database.windows.net`.

Preparing for performance degradation

A typical Azure application uses multiple Azure service and consists of multiple components. The automated failover of the failover group is triggered based on the state the Azure SQL components alone. Other Azure services in the primary region may not be affected by the outage and their components may still be available in that region. Once the primary databases switch to the DR region, the latency between the dependent components may increase. To avoid the impact of higher latency on the application's performance, ensure the redundancy of all the application's components in the DR region and follow these [network security guidelines](#).

Preparing for data loss

If an outage is detected, SQL automatically triggers read-write failover if there is zero data loss to the best of our knowledge. Otherwise, it waits for the period you specified by `GracePeriodWithDataLossHours`. If you specified `GracePeriodWithDataLossHours`, be prepared for data loss. In general, during outages, Azure favors availability. If you cannot afford data loss, make sure to set `GracePeriodWithDataLossHours` to a sufficiently large number, such as 24 hours.

The DNS update of the read-write listener will happen immediately after the failover is initiated. This operation will not result in data loss. However, the process of switching database roles can take up to 5 minutes under normal conditions. Until it is completed, some databases in the new primary instance will still be read-only. If failover is initiated using PowerShell, the entire operation is synchronous. If it is initiated using the Azure portal, the UI will indicate completion status. If it is initiated using the REST API, use standard Azure Resource Manager's polling mechanism to monitor for completion.

IMPORTANT

Use manual group failover to move primaries back to the original location. When the outage that caused the failover is mitigated, you can move your primary databases to the original location. To do that you should initiate the manual failover of the group.

Changing secondary region of the failover group

Let's assume that instance A is the primary instance, instance B is the existing secondary instance, and instance C is the new secondary instance in the third region. To make the transition, follow these steps:

1. Create instance C with same size as A and in the same DNS zone.
2. Delete the failover group between instances A and B. At this point the logins will be failing because the SQL aliases for the failover group listeners have been deleted and the gateway will not recognize the failover group name. The secondary databases will be disconnected from the primaries and will become read-write databases.
3. Create a failover group with the same name between instance A and C. Follow the instructions in [failover group with managed instance tutorial](#). This is a size-of-data operation and will complete when all databases from instance A are seeded and synchronized.
4. Delete instance B if not needed to avoid unnecessary charges.

NOTE

After step 2 and until step 3 is completed the databases in instance A will remain unprotected from a catastrophic failure of instance A.

Changing primary region of the failover group

Let's assume instance A is the primary instance, instance B is the existing secondary instance, and instance C is the new primary instance in the third region. To make the transition, follow these steps:

1. Create instance C with same size as B and in the same DNS zone.
2. Connect to instance B and manually failover to switch the primary instance to B. Instance A will become the new secondary instance automatically.
3. Delete the failover group between instances A and B. At this point the logins will be failing because the SQL aliases for the failover group listeners have been deleted and the gateway will not recognize the failover group name. The secondary databases will be disconnected from the primaries and will become read-write databases.
4. Create a failover group with the same name between instance A and C. Follow the instructions in the [failover group with managed instance tutorial](#). This is a size-of-data operation and will complete when all databases from instance A are seeded and synchronized.
5. Delete instance A if not needed to avoid unnecessary charges.

NOTE

After step 3 and until step 4 is completed the databases in instance A will remain unprotected from a catastrophic failure of instance A.

IMPORTANT

When the failover group is deleted, the DNS records for the listener endpoints are also deleted. At that point, there is a non-zero probability of somebody else creating a failover group or server alias with the same name, which will prevent you from using it again. To minimize the risk, don't use generic failover group names.

Failover groups and network security

For some applications the security rules require that the network access to the data tier is restricted to a specific component or components such as a VM, web service etc. This requirement presents some challenges for business continuity design and the use of the failover groups. Consider the following options when implementing such restricted access.

Using failover groups and virtual network rules

If you are using [Virtual Network service endpoints and rules](#) to restrict access to your SQL database, be aware that Each Virtual Network service endpoint applies to only one Azure region. The endpoint does not enable other regions to accept communication from the subnet. Therefore, only the client applications deployed in the same region can connect to the primary database. Since the failover results in the SQL client sessions being rerouted to a server in a different (secondary) region, these sessions will fail if originated from a client outside of that region. For that reason, the automatic failover policy cannot be enabled if the participating servers are included in the Virtual Network rules. To support manual failover, follow these steps:

1. Provision the redundant copies of the front-end components of your application (web service, virtual machines etc.) in the secondary region
2. Configure the [virtual network rules](#) individually for primary and secondary server
3. Enable the [front-end failover using a Traffic manager configuration](#)
4. Initiate manual failover when the outage is detected. This option is optimized for the applications that require consistent latency between the front-end and the data tier and supports recovery when either front end, data tier or both are impacted by the outage.

NOTE

If you are using the **read-only listener** to load-balance a read-only workload, make sure that this workload is executed in a VM or other resource in the secondary region so it can connect to the secondary database.

Using failover groups and SQL database firewall rules

If your business continuity plan requires failover using groups with automatic failover, you can restrict access to your SQL database using the traditional firewall rules. To support automatic failover, follow these steps:

1. [Create a public IP](#)
2. [Create a public load balancer](#) and assign the public IP to it.
3. [Create a virtual network and the virtual machines](#) for your front-end components
4. [Create network security group](#) and configure inbound connections.
5. Ensure that the outbound connections are open to Azure SQL database by using 'Sql' [service tag](#).
6. Create a [SQL database firewall rule](#) to allow inbound traffic from the public IP address you create in step 1.

For more information about on how to configure outbound access and what IP to use in the firewall rules, see [Load balancer outbound connections](#).

The above configuration will ensure that the automatic failover will not block connections from the front-end components and assumes that the application can tolerate the longer latency between the front end and the data tier.

IMPORTANT

To guarantee business continuity for regional outages you must ensure geographic redundancy for both front-end components and the databases.

Enabling geo-replication between managed instances and their VNets

When you set up a failover group between primary and secondary managed instances in two different

regions, each instance is isolated using an independent virtual network. To allow replication traffic between these VNets ensure these prerequisites are met:

1. The two managed instances need to be in different Azure regions.
2. The two managed instances need to be the same service tier, and have the same storage size.
3. Your secondary managed instance must be empty (no user databases).
4. The virtual networks used by the managed instances need to be connected through a [VPN Gateway](#) or [Express Route](#). When two virtual networks connect through an on-premises network, ensure there is no firewall rule blocking ports 5022, and 11000-11999. Global VNet Peering is not supported.
5. The two managed instance VNets cannot have overlapping IP addresses.
6. You need to set up your Network Security Groups (NSG) such that ports 5022 and the range 11000~12000 are open inbound and outbound for connections from the subnet of the other managed instance. This is to allow replication traffic between the instances.

IMPORTANT

Misconfigured NSG security rules leads to stuck database copy operations.

7. The secondary instance is configured with the correct DNS zone ID. DNS zone is a property of a managed instance and virtual cluster, and its ID is included in the host name address. The zone ID is generated as a random string when the first managed instance is created in each VNet and the same ID is assigned to all other instances in the same subnet. Once assigned, the DNS zone cannot be modified. Managed instances included in the same failover group must share the DNS zone. You accomplish this by passing the primary instance's zone ID as the value of `DnsZonePartner` parameter when creating the secondary instance.

NOTE

For a detailed tutorial on configuring failover groups with managed instance, see [add a managed instance to a failover group](#).

Upgrading or downgrading a primary database

You can upgrade or downgrade a primary database to a different compute size (within the same service tier, not between General Purpose and Business Critical) without disconnecting any secondary databases. When upgrading, we recommend that you upgrade all of the secondary databases first, and then upgrade the primary. When downgrading, reverse the order: downgrade the primary first, and then downgrade all of the secondary databases. When you upgrade or downgrade the database to a different service tier, this recommendation is enforced.

This sequence is recommended specifically to avoid the problem where the secondary at a lower SKU gets overloaded and must be re-seeded during an upgrade or downgrade process. You could also avoid the problem by making the primary read-only, at the expense of impacting all read-write workloads against the primary.

NOTE

If you created secondary database as part of the failover group configuration it is not recommended to downgrade the secondary database. This is to ensure your data tier has sufficient capacity to process your regular workload after failover is activated.

Preventing the loss of critical data

Due to the high latency of wide area networks, continuous copy uses an asynchronous replication mechanism. Asynchronous replication makes some data loss unavoidable if a failure occurs. However, some applications may require no data loss. To protect these critical updates, an application developer can call the `sp_wait_for_database_copy_sync` system procedure immediately after committing the transaction. Calling `sp_wait_for_database_copy_sync` blocks the calling thread until the last committed transaction has been transmitted to the secondary database. However, it does not wait for the transmitted transactions to be replayed and committed on the secondary. `sp_wait_for_database_copy_sync` is scoped to a specific continuous copy link. Any user with the connection rights to the primary database can call this procedure.

NOTE

`sp_wait_for_database_copy_sync` prevents data loss after failover, but does not guarantee full synchronization for read access. The delay caused by a `sp_wait_for_database_copy_sync` procedure call can be significant and depends on the size of the transaction log at the time of the call.

Failover groups and point-in-time restore

For information about using point-in-time restore with failover groups, see [Point in Time Recovery \(PITR\)](#).

Limitations of failover groups

Be aware of the following limitations:

- Failover group cannot be created between two servers or instances in the same Azure regions.
- Failover group cannot be renamed. You will need to delete the group and re-create it with a different name.
- Database rename is not supported for instances in failover group. You will need to temporarily delete failover group to be able to rename a database.

Programmatically managing failover groups

As discussed previously, auto-failover groups and active geo-replication can also be managed programmatically using Azure PowerShell and the REST API. The following tables describe the set of commands available. Active geo-replication includes a set of Azure Resource Manager APIs for management, including the [Azure SQL Database REST API](#) and [Azure PowerShell cmdlets](#). These APIs require the use of resource groups and support role-based security (RBAC). For more information on how to implement access roles, see [Azure Role-Based Access Control](#).

- [PowerShell](#)
- [Azure CLI](#)

Manage SQL database failover with single databases and elastic pools

CMDLET	DESCRIPTION
New-AzSqlDatabaseFailoverGroup	This command creates a failover group and registers it on both primary and secondary servers
Remove-AzSqlDatabaseFailoverGroup	Removes a failover group from the server
Get-AzSqlDatabaseFailoverGroup	Retrieves a failover group's configuration
Set-AzSqlDatabaseFailoverGroup	Modifies configuration of a failover group
Switch-AzSqlDatabaseFailoverGroup	Triggers failover of a failover group to the secondary server
Add-AzSqlDatabaseToFailoverGroup	Adds one or more databases to a failover group

Manage SQL database failover groups with managed instances

CMDLET	DESCRIPTION
New-AzSqlDatabaseInstanceFailoverGroup	This command creates a failover group and registers it on both primary and secondary instances
Set-AzSqlDatabaseInstanceFailoverGroup	Modifies configuration of a failover group
Get-AzSqlDatabaseInstanceFailoverGroup	Retrieves a failover group's configuration
Switch-AzSqlDatabaseInstanceFailoverGroup	Triggers failover of a failover group to the secondary instance
Remove-AzSqlDatabaseInstanceFailoverGroup	Removes a failover group

IMPORTANT

For a sample script, see [Configure and failover a failover group for a single database](#).

REST API: Manage SQL database failover groups with single and pooled databases

API	DESCRIPTION
Create or Update Failover Group	Creates or updates a failover group
Delete Failover Group	Removes a failover group from the server
Failover (Planned)	Triggers failover from the current primary server to the secondary server with full data synchronization.
Force Failover Allow Data Loss	Triggers failover from the current primary server to the secondary server without synchronizing data. This operation may result in data loss.
Get Failover Group	Retrieves a failover group's configuration.

API	DESCRIPTION
List Failover Groups By Server	Lists the failover groups on a server.
Update Failover Group	Updates a failover group's configuration.

REST API: Manage failover groups with Managed Instances

API	DESCRIPTION
Create or Update Failover Group	Creates or updates a failover group's configuration
Delete Failover Group	Removes a failover group from the instance
Failover (Planned)	Triggers failover from the current primary instance to this instance with full data synchronization.
Force Failover Allow Data Loss	Triggers failover from the current primary instance to the secondary instance without synchronizing data. This operation may result in data loss.
Get Failover Group	retrieves a failover group's configuration.
List Failover Groups - List By Location	Lists the failover groups in a location.

Next steps

- For detailed tutorials, see
 - [Add single database to a failover group](#)
 - [Add elastic pool to a failover group](#)
 - [Add a managed instance to a failover group](#)
- For sample scripts, see:
 - [Use PowerShell to configure active geo-replication for a single database in Azure SQL Database](#)
 - [Use PowerShell to configure active geo-replication for a pooled database in Azure SQL Database](#)
 - [Use PowerShell to add an Azure SQL Database single database to a failover group](#)
- For a business continuity overview and scenarios, see [Business continuity overview](#)
- To learn about Azure SQL Database automated backups, see [SQL Database automated backups](#).
- To learn about using automated backups for recovery, see [Restore a database from the service-initiated backups](#).
- To learn about authentication requirements for a new primary server and database, see [SQL Database security after disaster recovery](#).

Configure and manage Azure SQL Database security for geo-restore or failover

11/7/2019 • 4 minutes to read • [Edit Online](#)

This article describes the authentication requirements to configure and control [active geo-replication](#) and [auto-failover groups](#). It also provides the steps required to set up user access to the secondary database. Finally, it also describes how to enable access to the recovered database after using [geo-restore](#). For more information on recovery options, see [Business Continuity Overview](#).

Disaster recovery with contained users

Unlike traditional users, which must be mapped to logins in the master database, a contained user is managed completely by the database itself. This has two benefits. In the disaster recovery scenario, the users can continue to connect to the new primary database or the database recovered using geo-restore without any additional configuration, because the database manages the users. There are also potential scalability and performance benefits from this configuration from a login perspective. For more information, see [Contained Database Users - Making Your Database Portable](#).

The main trade-off is that managing the disaster recovery process at scale is more challenging. When you have multiple databases that use the same login, maintaining the credentials using contained users in multiple databases may negate the benefits of contained users. For example, the password rotation policy requires that changes be made consistently in multiple databases rather than changing the password for the login once in the master database. For this reason, if you have multiple databases that use the same user name and password, using contained users is not recommended.

How to configure logins and users

If you are using logins and users (rather than contained users), you must take extra steps to ensure that the same logins exist in the master database. The following sections outline the steps involved and additional considerations.

NOTE

It is also possible to use Azure Active Directory (AAD) logins to manage your databases. For more information, see [Azure SQL logins and users](#).

Set up user access to a secondary or recovered database

In order for the secondary database to be usable as a read-only secondary database, and to ensure proper access to the new primary database or the database recovered using geo-restore, the master database of the target server must have the appropriate security configuration in place before the recovery.

The specific permissions for each step are described later in this topic.

Preparing user access to a geo-replication secondary should be performed as part configuring geo-replication. Preparing user access to the geo-restored databases should be performed at any time when the original server is online (e.g. as part of the DR drill).

NOTE

If you fail over or geo-restore to a server that does not have properly configured logins, access to it will be limited to the server admin account.

Setting up logins on the target server involves three steps outlined below:

1. Determine logins with access to the primary database

The first step of the process is to determine which logins must be duplicated on the target server. This is accomplished with a pair of SELECT statements, one in the logical master database on the source server and one in the primary database itself.

Only the server admin or a member of the **LoginManager** server role can determine the logins on the source server with the following SELECT statement.

```
SELECT [name], [sid]
FROM [sys].[sql_logins]
WHERE [type_desc] = 'SQL_Login'
```

Only a member of the db_owner database role, the dbo user, or server admin, can determine all of the database user principals in the primary database.

```
SELECT [name], [sid]
FROM [sys].[database_principals]
WHERE [type_desc] = 'SQL_USER'
```

2. Find the SID for the logins identified in step 1

By comparing the output of the queries from the previous section and matching the SIDs, you can map the server login to database user. Logins that have a database user with a matching SID have user access to that database as that database user principal.

The following query can be used to see all of the user principals and their SIDs in a database. Only a member of the db_owner database role or server admin can run this query.

```
SELECT [name], [sid]
FROM [sys].[database_principals]
WHERE [type_desc] = 'SQL_USER'
```

NOTE

The **INFORMATION_SCHEMA** and **sys** users have *NULL* SIDs, and the **guest** SID is **0x00**. The **dbo** SID may start with *0x01060000000001648000000000048454*, if the database creator was the server admin instead of a member of **DbManager**.

3. Create the logins on the target server

The last step is to go to the target server, or servers, and generate the logins with the appropriate SIDs. The basic syntax is as follows.

```
CREATE LOGIN [<login name>]
WITH PASSWORD = <login password>,
SID = <desired login SID>
```

NOTE

If you want to grant user access to the secondary, but not to the primary, you can do that by altering the user login on the primary server by using the following syntax.

```
ALTER LOGIN <login name> DISABLE
```

DISABLE doesn't change the password, so you can always enable it if needed.

Next steps

- For more information on managing database access and logins, see [SQL Database security: Manage database access and login security](#).
- For more information on contained database users, see [Contained Database Users - Making Your Database Portable](#).
- To learn about active geo-replication, see [Active geo-replication](#).
- To learn about auto-failover groups, see [Auto-failover groups](#).
- For information about using geo-restore, see [geo-restore](#)

Restore an Azure SQL Database or failover to a secondary

11/7/2019 • 5 minutes to read • [Edit Online](#)

Azure SQL Database offers the following capabilities for recovering from an outage:

- [Active geo-replication](#)
- [Auto-failover groups](#)
- [Geo-restore](#)
- [Zone-redundant databases](#)

To learn about business continuity scenarios and the features supporting these scenarios, see [Business continuity](#).

NOTE

If you are using zone-redundant Premium or Business Critical databases or pools, the recovery process is automated and the rest of this material does not apply.

NOTE

Both primary and secondary databases are required to have the same service tier. It is also strongly recommended that the secondary database is created with the same compute size (DTUs or vCores) as the primary. For more information, see [Upgrading or downgrading as primary database](#).

NOTE

Use one or several failover groups to manage failover of multiple databases. If you add an existing geo-replication relationship to the failover group, make sure the geo-secondary is configured with the same service tier and compute size as the primary. For more information, see [Use auto-failover groups to enable transparent and coordinated failover of multiple databases](#).

Prepare for the event of an outage

For success with recovery to another data region using either failover groups or geo-redundant backups, you need to prepare a server in another data center outage to become the new primary server should the need arise as well as have well-defined steps documented and tested to ensure a smooth recovery. These preparation steps include:

- Identify the SQL Database server in another region to become the new primary server. For geo-restore, this is generally a server in the [paired region](#) for the region in which your database is located. This eliminates the additional traffic cost during the geo-restoring operations.
- Identify, and optionally define, the server-level IP firewall rules needed on for users to access the new primary database.
- Determine how you are going to redirect users to the new primary server, such as by changing connection strings or by changing DNS entries.
- Identify, and optionally create, the logins that must be present in the master database on the new primary server, and ensure these logins have appropriate permissions in the master database, if any. For more

information, see [SQL Database security after disaster recovery](#)

- Identify alert rules that need to be updated to map to the new primary database.
- Document the auditing configuration on the current primary database
- Perform a [disaster recovery drill](#). To simulate an outage for geo-restore, you can delete or rename the source database to cause application connectivity failure. To simulate an outage using failover groups, you can disable the web application or virtual machine connected to the database or failover the database to cause application connectivity failures.

When to initiate recovery

The recovery operation impacts the application. It requires changing the SQL connection string or redirection using DNS and could result in permanent data loss. Therefore, it should be done only when the outage is likely to last longer than your application's recovery time objective. When the application is deployed to production you should perform regular monitoring of the application health and use the following data points to assert that the recovery is warranted:

1. Permanent connectivity failure from the application tier to the database.
2. The Azure portal shows an alert about an incident in the region with broad impact.

NOTE

If you are using failover groups and chose automatic failover, the recovery process is automated and transparent to the application.

Depending on your application tolerance to downtime and possible business liability you can consider the following recovery options.

Use the [Get Recoverable Database \(LastAvailableBackupDate\)](#) to get the latest Geo-replicated restore point.

Wait for service recovery

The Azure teams work diligently to restore service availability as quickly as possible but depending on the root cause it can take hours or days. If your application can tolerate significant downtime you can simply wait for the recovery to complete. In this case, no action on your part is required. You can see the current service status on our [Azure Service Health Dashboard](#). After the recovery of the region, your application's availability is restored.

Fail over to geo-replicated secondary server in the failover group

If your application's downtime can result in business liability, you should be using failover groups. It enables the application to quickly restore availability in a different region in case of an outage. For a tutorial, see [Implement a geo-distributed database](#).

To restore availability of the database(s) you need to initiate the failover to the secondary server using one of the supported methods.

Use one of the following guides to fail over to a geo-replicated secondary database:

- [Fail over to a geo-replicated secondary server using the Azure portal](#)
- [Fail over to the secondary server using PowerShell](#)
- [Fail over to a secondary server using Transact-SQL \(T-SQL\)](#)

Recover using geo-restore

If your application's downtime does not result in business liability you can use [geo-restore](#) as a method to recover

your application database(s). It creates a copy of the database from its latest geo-redundant backup.

Configure your database after recovery

If you are using geo-restore to recover from an outage, you must make sure that the connectivity to the new databases is properly configured so that the normal application function can be resumed. This is a checklist of tasks to get your recovered database production ready.

Update connection strings

Because your recovered database resides in a different server, you need to update your application's connection string to point to that server.

For more information about changing connection strings, see the appropriate development language for your [connection library](#).

Configure Firewall Rules

You need to make sure that the firewall rules configured on server and on the database match those that were configured on the primary server and primary database. For more information, see [How to: Configure Firewall Settings \(Azure SQL Database\)](#).

Configure logins and database users

You need to make sure that all the logins used by your application exist on the server which is hosting your recovered database. For more information, see [Security Configuration for geo-replication](#).

NOTE

You should configure and test your server firewall rules and logins (and their permissions) during a disaster recovery drill. These server-level objects and their configuration may not be available during the outage.

Setup telemetry alerts

You need to make sure your existing alert rule settings are updated to map to the recovered database and the different server.

For more information about database alert rules, see [Receive Alert Notifications](#) and [Track Service Health](#).

Enable auditing

If auditing is required to access your database, you need to enable Auditing after the database recovery. For more information, see [Database auditing](#).

Next steps

- To learn about Azure SQL Database automated backups, see [SQL Database automated backups](#)
- To learn about business continuity design and recovery scenarios, see [Continuity scenarios](#)
- To learn about using automated backups for recovery, see [restore a database from the service-initiated backups](#)

Performing Disaster Recovery Drill

11/7/2019 • 2 minutes to read • [Edit Online](#)

It is recommended that validation of application readiness for recovery workflow is performed periodically. Verifying the application behavior and implications of data loss and/or the disruption that failover involves is a good engineering practice. It is also a requirement by most industry standards as part of business continuity certification.

Performing a disaster recovery drill consists of:

- Simulating data tier outage
- Recovering
- Validate application integrity post recovery

Depending on how you [designed your application for business continuity](#), the workflow to execute the drill can vary. This article describes the best practices for conducting a disaster recovery drill in the context of Azure SQL Database.

Geo-restore

To prevent the potential data loss when conducting a disaster recovery drill, perform the drill using a test environment by creating a copy of the production environment and using it to verify the application's failover workflow.

Outage simulation

To simulate the outage, you can rename the source database. This name change causes application connectivity failures.

Recovery

- Perform the geo-restore of the database into a different server as described [here](#).
- Change the application configuration to connect to the recovered database and follow the [Configure a database after recovery](#) guide to complete the recovery.

Validation

Complete the drill by verifying the application integrity post recovery (including connection strings, logins, basic functionality testing, or other validations part of standard application signoffs procedures).

Failover groups

For a database that is protected using failover groups, the drill exercise involves planned failover to the secondary server. The planned failover ensures that the primary and the secondary databases in the failover group remain in sync when the roles are switched. Unlike the unplanned failover, this operation does not result in data loss, so the drill can be performed in the production environment.

Outage simulation

To simulate the outage, you can disable the web application or virtual machine connected to the database. This outage simulation results in the connectivity failures for the web clients.

Recovery

- Make sure the application configuration in the DR region points to the former secondary, which becomes the fully accessible new primary.

- Initiate [planned failover](#) of the failover group from the secondary server.
- Follow the [Configure a database after recovery](#) guide to complete the recovery.

Validation

Complete the drill by verifying the application integrity post recovery (including connectivity, basic functionality testing, or other validations required for the drill signoffs).

Next steps

- To learn about business continuity scenarios, see [Continuity scenarios](#).
- To learn about Azure SQL Database automated backups, see [SQL Database automated backups](#)
- To learn about using automated backups for recovery, see [restore a database from the service-initiated backups](#).
- To learn about faster recovery options, see [Active geo-replication](#) and [Auto-failover groups](#).

Configure a failover group for Azure SQL Database

2/18/2020 • 16 minutes to read • [Edit Online](#)

This topic teaches you how to configure an [auto-failover group](#) for an Azure SQL Database single database, elastic pool, and managed instance using the Azure portal, or PowerShell.

Single database

Create the failover group and add a single database to it using the Azure portal, or PowerShell.

Prerequisites

Consider the following prerequisites:

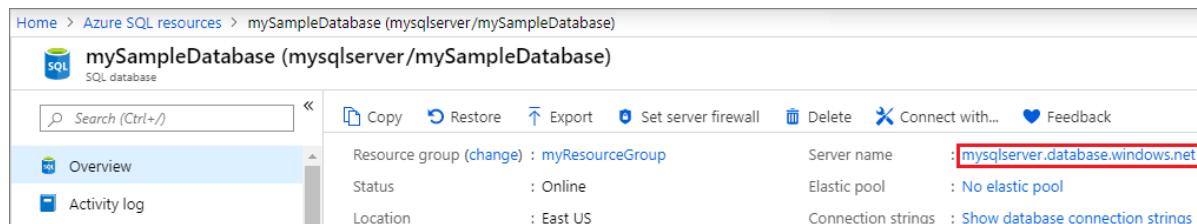
- The server login and firewall settings for the secondary server must match that of your primary server.

Create failover group

- [Portal](#)
- [PowerShell](#)

Create your failover group and add your single database to it using the Azure portal.

- Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
- Select the single database you want to add to the failover group.
- Select the name of the server under **Server name** to open the settings for the server.



The screenshot shows the Azure portal interface for managing a SQL database. The top navigation bar includes 'Home', 'Azure SQL resources', and the specific resource name 'mySampleDatabase (mysqlserver/mySampleDatabase)'. Below the navigation is a search bar and a toolbar with actions like 'Copy', 'Restore', 'Export', 'Set server firewall', 'Delete', 'Connect with...', and 'Feedback'. A sidebar on the left lists 'Overview' and 'Activity log'. The main content area displays resource details: 'Resource group (change) : myResourceGroup', 'Status : Online', 'Location : East US', 'Server name : mysqlserver.database.windows.net' (with a red box around it), 'Elastic pool : No elastic pool', and 'Connection strings : Show database connection strings'.

- Select **Failover groups** under the **Settings** pane, and then select **Add group** to create a new failover group.

mysqlsample - Failover groups

Add group Refresh

Failover group are a SQL server feature designed to automatically manage repl

NAME	PRIMARY SERVER	SECOND
You have no group created		

- On the **Failover Group** page, enter or select the required values, and then select **Create**.

- Databases within the group:** Choose the database you want to add to your failover group. Adding the database to the failover group will automatically start the geo-replication process.

Failover group

Databases for failover group

Select all Selected/Eligible databases 0/1

NAME	ROLE	SECONDARY SERVER	STATUS
mySampleDatabase			Online

Summary
Databases on secondary (excluding ones in Elastic Pools)
Elastic Pools on secondary server

Monthly cost

Test failover

Test failover of your failover group using the Azure portal, or PowerShell.

- [Portal](#)
- [PowerShell](#)

Test failover of your failover group using the Azure portal.

- Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
- Select the single database you want to add to the failover group.

mySampleDatabase (mysqlserver/mySampleDatabase)

Overview

Resource group (change) : myResourceGroup Server name : mysqlserver.database.windows.net

Status : Online Elastic pool : No elastic pool

Location : East US Connection strings : Show database connection strings

- Select **Failover groups** under the **Settings** pane and then choose the failover group you just created.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase) > mysqlserver - Failover groups

mysqlserver - Failover groups

SQL server

Search (Ctrl+ /) < + Add group Refresh

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Settings
- Quick start
- Failover groups**
- Manage Backups
- Active Directory admin
- SQL databases
- SQL elastic pools
- Deleted databases

Failover group are a SQL server feature designed to automatically manage

NAME	PRIMARY SERVER	SECONDARY SERVER
failovergrouptutorial	mysqlserver	mysqlsecondary

- Review which server is primary and which server is secondary.
- Select **Failover** from the task pane to fail over your failover group containing your single database.
- Select **Yes** on the warning that notifies you that TDS sessions will be disconnected.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase) > mysqlserver - Failover groups

failovergrouptutorial

failover-sqlserver

Save Discard Add databases Edit configuration Remove databases **Failover** Forced Failover Delete

Configuration details Databases within group Databases selected to be added (0) Databases selected for removal (0)

SERVER	ROLE	READ/WRITE FAILOVER POLICY
mysqlserver (West US)	Primary	Automatic
mysqlsecondary (East US)	Secondary	

- Review which server is now primary and which server is secondary. If failover succeeded, the two servers should have swapped roles.
- Select **Failover** again to fail the servers back to their originally roles.

IMPORTANT

If you need to delete the secondary database, remove it from the failover group before deleting it. Deleting a secondary database before it is removed from the failover group can cause unpredictable behavior.

Elastic pool

Create the failover group and add an elastic pool to it using the Azure portal, or PowerShell.

Prerequisites

Consider the following prerequisites:

- The server login and firewall settings for the secondary server must match that of your primary server.

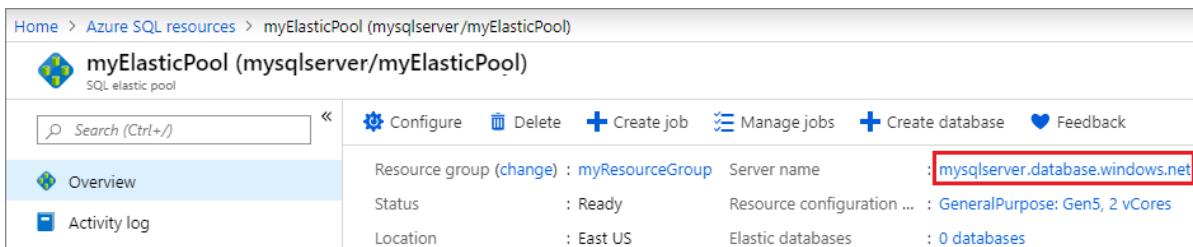
Create the failover group

Create the failover group for your elastic pool using the Azure portal, or PowerShell.

- [Portal](#)
- [PowerShell](#)

Create your failover group and add your elastic pool to it using the Azure portal.

- Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
- Select the elastic pool you want to add to the failover group.
- On the **Overview** pane, select the name of the server under **Server name** to open the settings for the server.



Home > Azure SQL resources > myElasticPool (mysqlserver/myElasticPool)

myElasticPool (mysqlserver/myElasticPool)
SQL elastic pool

Search (Ctrl+ /)	Configure	Delete	Create job	Manage jobs	Create database	Feedback
Overview	Resource group (change) : myResourceGroup	Server name	: mysqlserver.database.windows.net			
Activity log	Status : Ready	Resource configuration ...	: GeneralPurpose: Gen5, 2 vCores			
	Location : East US	Elastic databases	: 0 databases			

- Select **Failover groups** under the **Settings** pane, and then select **Add group** to create a new failover group.

The screenshot shows the 'Failover groups' page for the 'mysqlserver' resource. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Quick start, and Failover groups. The 'Failover groups' link is highlighted with a red box. At the top right, there's a search bar, a 'Add group' button (also highlighted with a red box), and a 'Refresh' button. The main area displays a table with columns 'NAME', 'PRIMARY SERVER', and 'SECOND'. A message says 'You have no group created'. Below the table, there's a note: 'Failover group are a SQL server feature designed to automatically manage repl'.

- On the **Failover Group** page, enter or select the required values, and then select **Create**. Either create a new secondary server, or select an existing secondary server.
- Select **Databases within the group** then choose the elastic pool you want to add to the failover group. If an elastic pool does not already exist on the secondary server, a warning appears prompting you to create an elastic pool on the secondary server. Select the warning, and then select **OK** to create the elastic pool on the secondary server.

The screenshot shows the 'Databases' section of the 'Failover group' creation page. It includes fields for 'Failover group name' (set to 'failovergroupTutorial'), 'Secondary server' (set to 'mysqlsecondary (East US)'), and 'Read/Write failover policy' (set to 'Automatic'). A red box highlights the 'Database within the group' section, which contains a link 'Select databases to add' with the number '1'. To the right, there's a 'Databases' table with one row selected: 'mySampleDatabase' (Primary, mysqlsecondary, Online). A red box highlights a warning message: 'As myElasticPool does not exist on mysqlsecondary, you need to create it by clicking here.' Below the table is a 'Summary' section. To the right, there's an 'Elastic pool' configuration panel with fields for 'Name' (set to 'myElasticPool'), 'Subscription', 'Resource group' (set to 'myResourceGroup'), 'Server' (set to 'mysqlsecondary (East US)'), 'Pricing tier' (set to 'GeneralPurpose Pool'), and 'Configured' (set to '1 eDTU, 0 databases'). At the bottom right of the configuration panel is a red box highlighting the 'OK' button.

- Select **Select** to apply your elastic pool settings to the failover group, and then select **Create** to create your failover group. Adding the elastic pool to the failover group will automatically start the geo-replication process.

Test failover

Test failover of your elastic pool using the Azure portal, or PowerShell.

- Portal
- PowerShell

Fail your failover group over to the secondary server, and then fail back using the Azure portal.

- Select **Azure SQL** in the left-hand menu of the **Azure portal**. If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
- Select the elastic pool you want to add to the failover group.

3. On the **Overview** pane, select the name of the server under **Server name** to open the settings for the server.

Home > Azure SQL resources > myElasticPool (mysqlserver/myElasticPool)

myElasticPool (mysqlserver/myElasticPool)

SQL elastic pool

Search (Ctrl+ /) <

Configure Delete + Create job Manage jobs + Create database Feedback

Resource group (change) : myResourceGroup Server name : mysqlserver.database.windows.net

Overview Status : Ready Resource configuration ... : GeneralPurpose: Gen5, 2 vCores

Activity log Location : East US Elastic databases : 0 databases

4. Select **Failover groups** under the **Settings** pane and then choose the failover group you created in section 2.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase) > mysqlserver - Failover groups

mysqlserver - Failover groups

SQL server

Search (Ctrl+ /) < + Add group Refresh

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings Quick start Failover groups Manage Backups Active Directory admin SQL databases SQL elastic pools Deleted databases

Failover group are a SQL server feature designed to automatically manage

NAME	PRIMARY SERVER	SECONDARY SERVER
failovergrouptutorial	mysqlserver	mysqlsecondary

5. Review which server is primary, and which server is secondary.
6. Select **Failover** from the task pane to fail over your failover group containing your elastic pool.
7. Select **Yes** on the warning that notifies you that TDS sessions will be disconnected.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase) > mysqlserver - Failover groups

failovergrouptutorial
failover-sqlserver

Save Discard Add databases Edit configuration Remove databases **Failover** Forced Failover Delete

Configuration details Databases within group Databases selected to be added (0) Databases selected for removal (0)

SERVER	ROLE	READ/WRITE FAILOVER POLICY
mysqlserver (West US)	Primary	Automatic
mysqlsecondary (East US)	Secondary	

8. Review which server is primary, which server is secondary. If failover succeeded, the two servers should have swapped roles.
9. Select **Failover** again to fail the failover group back to the original settings.

IMPORTANT

If you need to delete the secondary database, remove it from the failover group before deleting it. Deleting a secondary database before it is removed from the failover group can cause unpredictable behavior.

Managed instance

Create a failover group between two managed instances using the Azure portal, or PowerShell.

You will either need to configure [ExpressRoute](#) or to create a gateway for the virtual network of each managed instance, connect the two gateways, and then create the failover group.

Prerequisites

Consider the following prerequisites:

- The secondary managed instance must be empty.
- The subnet range for the secondary virtual network must not overlap the subnet range of the primary virtual network.
- The collation and timezone of the secondary instance must match that of the primary instance.
- When connecting the two gateways, the **Shared Key** should be the same for both connections.

Create primary virtual network gateway

If you have not configured [ExpressRoute](#), you can create the primary virtual network gateway with the Azure portal, or PowerShell.

- [Portal](#)
- [PowerShell](#)

Create the primary virtual network gateway using the Azure portal.

1. In the [Azure portal](#), go to your resource group and select the **Virtual network** resource for your primary managed instance.
2. Select **Subnets** under **Settings** and then select to add a new **Gateway subnet**. Leave the default values.

The screenshot shows the 'Subnets' blade for a virtual network named 'vnet-sql-mi-primary'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Address space, Connected devices, and Subnets. The 'Subnets' link is highlighted with a red box. The main area shows a table with one row labeled 'default'. At the top right, there is a 'Gateway subnet' button, which is also highlighted with a red box. To the right of the table, there is a 'Add subnet' dialog box with fields for Name (GatewaySubnet), Address range (CIDR block) (10.0.1.0/24), Network security group (None), Route table (None), Service endpoints (0 selected), and Subnet delegation (None).

3. Once the subnet gateway is created, select **Create a resource** from the left navigation pane and then type **Virtual network gateway** in the search box. Select the **Virtual network gateway** resource published by **Microsoft**.

The screenshot shows the Azure Marketplace search results for 'virtual network gateway'. The search bar contains 'virtual network gateway'. On the left, the navigation pane shows 'Create a resource' highlighted with a red box, followed by a list of services including Home, Dashboard, All services, Resource groups, SQL virtual machines, SQL servers, SQL managed instances, Virtual machines, All resources, Recent, App Services, and Virtual machines (classic). The search results show two cards: 'Virtual network gateway' by Microsoft, which is highlighted with a red box, and 'Virtual Network' by Microsoft.

4. Fill out the required fields to configure the gateway for your primary managed instance.

The following table shows the values necessary for the gateway for the primary managed instance:

FIELD	VALUE
Subscription	The subscription where your primary managed instance is.
Name	The name for your virtual network gateway.
Region	The region where your secondary managed instance is.
Gateway type	Select VPN .
VPN Type	Select Route-based
SKU	Leave default of VpnGw1 .
Location	The location where your secondary managed instance and secondary virtual network is.
Virtual network	Select the virtual network for your secondary managed instance.
Public IP address	Select Create new .
Public IP address name	Enter a name for your IP address.

5. Leave the other values as default, and then select **Review + create** to review the settings for your virtual network gateway.

Home > New > Marketplace > Virtual network gateway > Create virtual network gateway

Create virtual network gateway

* Subscription

Resource group myResourceGroup (derived from virtual network's resource group)

Instance details

* Name primary-mi-gateway ✓

* Region (US) West US ✓

* Gateway type VPN ExpressRoute

* VPN type Route-based Policy-based

* SKU VpnGw1 ✓

i Only virtual networks in the currently selected subscription and region are listed.

VIRTUAL NETWORK

* Virtual network vnet-sql-mi-primary ✓
[Create new](#)

Gateway subnet address range 10.0.1.0/24

Public IP address

* Public IP address Create new Use existing

* Public IP address name primary-gateway-IP ✓

Public IP address SKU Basic

* Assignment Dynamic Static

* Enable active-active mode Enabled Disabled

* Configure BGP ASN Enabled Disabled

6. Select **Create** to create your new virtual network gateway.

Create secondary virtual network gateway

Create the secondary virtual network gateway using the Azure portal, or PowerShell.

- [Portal](#)
- [PowerShell](#)

Repeat the steps in the previous section to create the virtual network subnet and gateway for the secondary managed instance. Fill out the required fields to configure the gateway for your secondary managed instance.

The following table shows the values necessary for the gateway for the secondary managed instance:

FIELD	VALUE
Subscription	The subscription where your secondary managed instance is.
Name	The name for your virtual network gateway, such as <input type="text"/> secondary-mi-gateway.
Region	The region where your secondary managed instance is.

FIELD	VALUE
Gateway type	Select VPN .
VPN Type	Select Route-based
SKU	Leave default of <code>VpnGw1</code> .
Location	The location where your secondary managed instance and secondary virtual network is.
Virtual network	Select the virtual network that was created in section 2, such as <code>vnet-sql-mi-secondary</code> .
Public IP address	Select Create new .
Public IP address name	Enter a name for your IP address, such as <code>secondary-gateway-IP</code> .

Create virtual network gateway

* Subscription	<input type="text"/>	▼
Resource group <small>?</small>	myResourceGroup (derived from virtual network's resource group)	
Instance details		
* Name	<input type="text"/> secondary-mi-gateway ✓	
* Region	<input type="text"/> (US) East US ▼	
* Gateway type <small>?</small>	<input checked="" type="radio"/> VPN <input type="radio"/> ExpressRoute	
* VPN type <small>?</small>	<input checked="" type="radio"/> Route-based <input type="radio"/> Policy-based	
* SKU <small>?</small>	<input type="text"/> VpnGw1 ▼	
<small>Only virtual networks in the currently selected subscription and region are listed.</small>		
VIRTUAL NETWORK		
* Virtual network <small>?</small>	<input type="text"/> vnet-sql-mi-secondary ▼	
Create new		
Gateway subnet address range	10.128.1.0/24	
Public IP address		
* Public IP address <small>?</small>	<input checked="" type="radio"/> Create new <input type="radio"/> Use existing	
* Public IP address name	<input type="text"/> secondary-gateway-IP ✓	
Public IP address SKU	Basic	
* Assignment	<input checked="" type="radio"/> Dynamic <input type="radio"/> Static	
* Enable active-active mode <small>?</small>	<input type="radio"/> Enabled <input checked="" type="radio"/> Disabled	
* Configure BGP ASN <small>?</small>	<input type="radio"/> Enabled <input checked="" type="radio"/> Disabled	

Connect the gateways

Create connections between the two gateways using the Azure portal, or PowerShell.

Two connections need to be created - the connection from the primary gateway to the secondary gateway, and then the connection from the secondary gateway to the primary gateway.

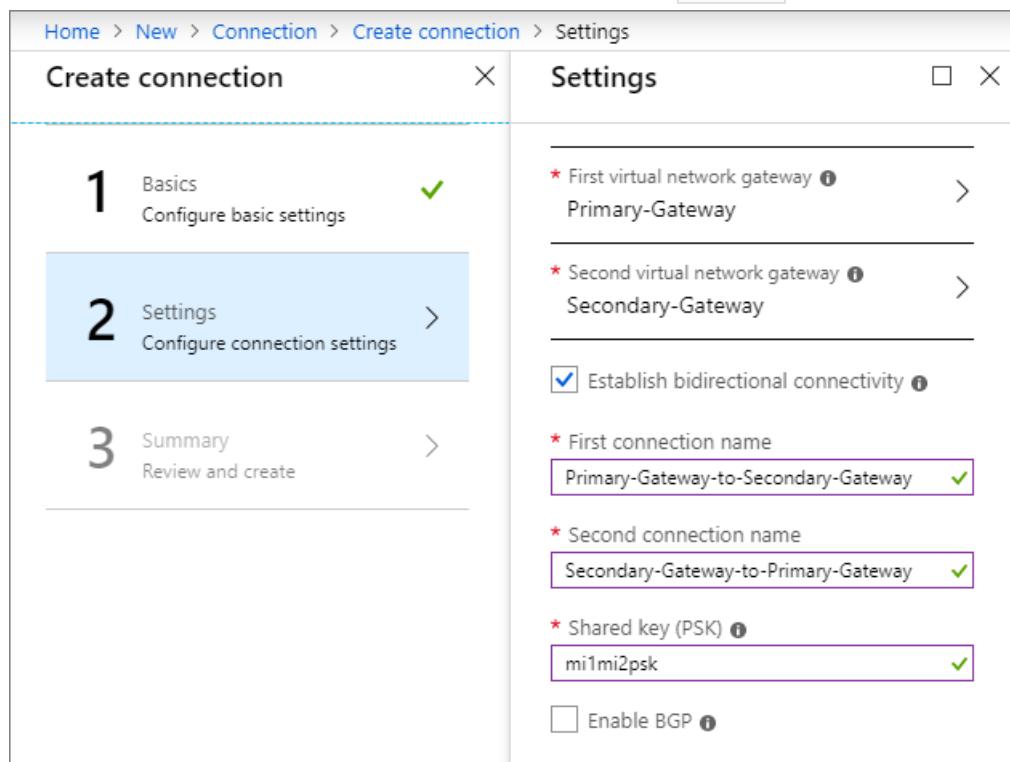
The shared key used for both connections should be the same for each connection.

- [Portal](#)
- [PowerShell](#)

Create connections between the two gateways using the Azure portal.

1. Select **Create a resource** from the [Azure portal](#).
2. Type `connection` in the search box and then press enter to search, which takes you to the **Connection** resource, published by Microsoft.
3. Select **Create** to create your connection.
4. On the **Basics** tab, select the following values and then select **OK**.

- a. Select **VNet-to-VNet** for the **Connection type**.
 - b. Select your subscription from the drop-down.
 - c. Select the resource group for your managed instance in the drop-down.
 - d. Select the location of your primary managed instance from the drop-down
5. On the **Settings** tab, select or enter the following values and then select **OK**:
- a. Choose the primary network gateway for the **First virtual network gateway**, such as **Primary-Gateway**.
 - b. Choose the secondary network gateway for the **Second virtual network gateway**, such as **Secondary-Gateway**.
 - c. Select the checkbox next to **Establish bidirectional connectivity**.
 - d. Either leave the default primary connection name, or rename it to a value of your choice.
 - e. Provide a **Shared key (PSK)** for the connection, such as **mi1m2psk**.



6. On the **Summary** tab, review the settings for your bidirectional connection and then select **OK** to create your connection.

Create the failover group

Create the failover group for your managed instances using the Azure portal, or PowerShell.

- [Portal](#)
- [PowerShell](#)

Create the failover group for your managed instances using Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select the primary managed instance you want to add to the failover group.
3. Under **Settings**, navigate to **Instance Failover Groups** and then choose to **Add group** to open the **Instance Failover Group** page.

Microsoft Azure

Home > Azure SQL resources > sql-mi-primary - Instance Failover Groups (preview)

sql-mi-primary - Instance Failover Groups (preview)

SQL managed instance

Add group

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Quick start

Connection strings

Active Directory admin

Pricing tier

Instance Failover Groups

The screenshot shows the Microsoft Azure portal interface for managing instance failover groups. On the left, a sidebar lists various Azure services. In the main content area, the 'Instance Failover Groups' page is displayed. A red box highlights the 'Add group' button at the top right. Another red box highlights the 'Instance Failover Groups' link in the sidebar. To the right of the sidebar is a world map with several regions marked by blue and green checkboxes, indicating where failover groups can be created.

- On the **Instance Failover Group** page, type the name of your failover group and then choose the secondary managed instance from the drop-down. Select **Create** to create your failover group.

Microsoft Azure

Home > SQL managed instances > sql-mi-primary

Instance Failover Group

Create a failover group to manage the automatic failover of managed instances.

Primary managed instance
sql-mi-primary (westus/General...)

* Failover group name
failovergrouptutorial

Secondary managed instance
sql-mi-secondary (eastus/GeneralPurpose)

Read/Write failover policy
Automatic

Read/Write grace period (hours)
1 hours

All databases within your primary managed instance will be replicated to your secondary managed instance.

- Once failover group deployment is complete, you will be taken back to the **Failover group** page.

Test failover

Test failover of your failover group using the Azure portal, or PowerShell.

- [Portal](#)
- [PowerShell](#)

Test failover of your failover group using the Azure portal.

1. Navigate to your *secondary* managed instance within the [Azure portal](#) and select **Instance Failover Groups** under settings.
2. Review which managed instance is the primary, and which managed instance is the secondary.
3. Select **Failover** and then select **Yes** on the warning about TDS sessions being disconnected.

NAME	PRIMARY MANAGED INSTANCE	SECONDARY MANAGED INSTANCE	READ/WRITE FAILOVER POLICY
failovergrouptutorial	<input checked="" type="checkbox"/> sql-mi-primary (westus)	<input checked="" type="checkbox"/> sql-mi-secondary (eastus)	Automatic

4. Review which managed instance is the primary and which instance is the secondary. If failover succeeded, the two instances should have switched roles.

NAME	PRIMARY MANAGED INSTANCE	SECONDARY MANAGED INSTANCE	READ/WRITE FAILOVER POLICY
failovergrouptutorial	<input checked="" type="checkbox"/> sql-mi-secondary (eastus)	<input checked="" type="checkbox"/> sql-mi-primary (westus)	Automatic

5. Go to the new *secondary* managed instance and select **Failover** once again to fail the primary instance back to the primary role.

Locate listener endpoint

Once your failover group is configured, update the connection string for your application to the listener endpoint. This will keep your application connected to the failover group listener, rather than the primary database, elastic pool, or managed instance. That way, you don't have to manually update the connection string every time your Azure SQL database entity fails over, and traffic is routed to whichever entity is currently primary.

The listener endpoint is in the form of `fog-name.database.windows.net`, and is visible in the Azure portal, when viewing the failover group:

failovergrouptutorial
mysqlsecondary

Save Discard Add databases Edit configuration Remove databases Failover

SERVER	ROLE
mysqlserver (East US)	Primary
mysqlsecondary (West US)	Secondary

Read/write listener endpoint

failovergrouptutorial.database.windows.net

Read-only listener endpoint

failovergrouptutorial.secondary.database.windows.net

Remarks

- Removing a failover group for a single or pooled database does not stop replication, and it does not delete the replicated database. You will need to manually stop geo-replication and delete the database from the secondary server if you want to add a single or pooled database back to a failover group after it's been removed. Failing to do either thing may result in an error similar to `The operation cannot be performed due to multiple errors` when attempting to add the database to the failover group.

Next steps

For detailed steps configuring a failover group, see the following tutorials:

- [Add a single database to a failover group](#)
- [Add an elastic pool to a failover group](#)
- [Add managed instances to a failover group](#)

For an overview of Azure SQL Database high availability options, see [geo-replication](#) and [auto-failover groups](#).

Monitoring and performance tuning

11/7/2019 • 16 minutes to read • [Edit Online](#)

Azure SQL Database provides tools and methods you can use to monitor usage easily, add or remove resources (such as CPU, memory, or I/O), troubleshoot potential problems, and make recommendations to improve the performance of a database. Features in Azure SQL Database can automatically fix problems in the databases.

Automatic tuning enables a database to adapt to the workload and automatically optimize performance. However, some custom issues might need troubleshooting. This article explains some best practices and some tools you can use to troubleshoot performance problems.

To ensure that a database runs without problems, you should:

- [Monitor database performance](#) to make sure that the resources assigned to the database can handle the workload. If the database is hitting resource limits, consider:
 - Identifying and optimizing the top resource-consuming queries.
 - Adding more resources by [upgrading the service tier](#).
- [Troubleshoot performance problems](#) to identify why a potential problem occurred and to identify the root cause of the problem. After you identify the root cause, take steps to fix the problem.

Monitor database performance

To monitor the performance of a SQL database in Azure, start by monitoring the resources used relative to the level of database performance you chose. Monitor the following resources:

- **CPU usage:** Check to see if the database is reaching 100 percent of CPU usage for an extended period of time. High CPU usage might indicate that you need to identify and tune queries that use the most compute power. High CPU usage might also indicate that the database or instance should be upgraded to a higher service tier.
- **Wait statistics:** Use [sys.dm_os_wait_stats \(Transact-SQL\)](#) to determine how long queries are waiting. Queries can be waiting on resources, queue waits, or external waits.
- **IO usage:** Check to see if the database is reaching the IO limits of the underlying storage.
- **Memory usage:** The amount of memory available for the database or instance is proportional to the number of vCores. Make sure the memory is enough for the workload. Page life expectancy is one of the parameters that can indicate how quickly the pages are removed from the memory.

The Azure SQL Database service includes tools and resources to help you troubleshoot and fix potential performance problems. You can identify opportunities to improve and optimize query performance without changing resources by reviewing [performance tuning recommendations](#).

Missing indexes and poorly optimized queries are common reasons for poor database performance. You can apply tuning recommendations to improve the performance of the workload. You can also let Azure SQL Database [automatically optimize performance of the queries](#) by applying all identified recommendations. Then verify that the recommendations improved database performance.

NOTE

Indexing is available in single database and elastic pools only. Indexing isn't available in a managed instance.

Choose from the following options to monitor and troubleshoot database performance:

- In the [Azure portal](#), select **SQL databases** and select the database. In the **Monitoring** chart, look for resources

approaching their maximum utilization. DTU consumption is shown by default. Select **Edit** to change the time range and values shown.

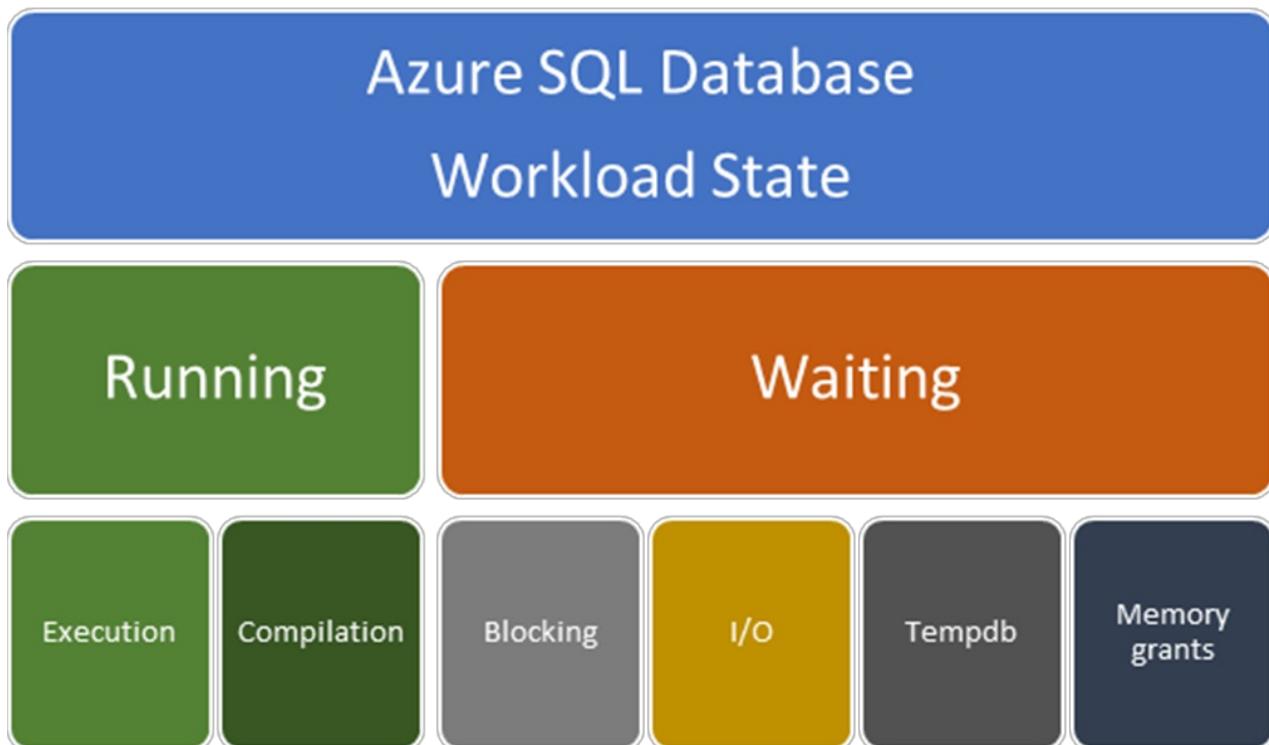
- Tools such as SQL Server Management Studio provide many useful reports, like [Performance Dashboard](#). Use these reports to monitor resource usage and identify top resource-consuming queries. You can use [Query Store](#) to identify queries whose performance has regressed.
- In the [Azure portal](#), use [Query Performance Insight](#) to identify the queries that use the most resources. This feature is available in single database and elastic pools only.
- Use [SQL Database Advisor](#) to view recommendations to help you create and drop indexes, parameterize queries, and fix schema problems. This feature is available in single database and elastic pools only.
- Use [Azure SQL Intelligent Insights](#) to automatically monitor database performance. When a performance problem is detected, a diagnostic log is generated. The log provides details and a root cause analysis (RCA) of the problem. A performance-improvement recommendation is provided when possible.
- [Enable automatic tuning](#) to let Azure SQL Database automatically fix performance problems.
- Use [dynamic management views \(DMVs\)](#), [extended events](#), and [Query Store](#) for help with more detailed troubleshooting of performance problems.

TIP

After you identify a performance problem, check out our [performance guidance](#) to find techniques to improve the performance of Azure SQL Database.

Troubleshoot performance problems

To diagnose and resolve performance problems, begin by finding out the state of each active query and the conditions that cause performance problems relevant to each workload state. To improve Azure SQL Database performance, you need to understand that each active query request from the application is in either a running state or a waiting state. As you troubleshoot a performance problem in Azure SQL Database, keep the following diagram in mind.



A performance problem in a workload can be caused by CPU contention (a *running-related* condition) or individual queries that are waiting on something (a *waiting-related* condition).

Running-related problems might be caused by:

- **Compilation problems:** SQL Query Optimizer might produce a suboptimal plan because of stale statistics, an incorrect estimate of the number of rows to be processed, or an inaccurate estimate of required memory. If you know the query was executed faster in the past or on another instance (either a managed instance or a SQL Server instance), compare the actual execution plans to see if they're different. Try to apply query hints or rebuild statistics or indexes to get the better plan. Enable automatic plan correction in Azure SQL Database to automatically mitigate these problems.
- **Execution problems:** If the query plan is optimal, it's probably hitting the database's resource limits, such as log write throughput. Or it might be using fragmented indexes that should be rebuilt. Execution problems can also happen when a large number of concurrent queries need the same resources. *Waiting-related* problems are usually related to execution problems, because the queries that don't execute efficiently are probably waiting for some resources.

Waiting-related problems might be caused by:

- **Blocking:** One query might hold the lock on objects in the database while others try to access the same objects. You can identify blocking queries by using DMVs or monitoring tools.
- **IO problems:** Queries might be waiting for the pages to be written to the data or log files. In this case, check the `INSTANCE_LOG_RATE_GOVERNOR`, `WRITE_LOG`, or `PAGEIOLATCH_*` wait statistics in the DMV.
- **TempDB problems:** If the workload uses temporary tables or there are TempDB spills in the plans, the queries might have a problem with TempDB throughput.
- **Memory-related problems:** If the workload doesn't have enough memory, the page life expectancy might drop, or the queries might get less memory than they need. In some cases, built-in intelligence in Query Optimizer will fix memory-related problems.

The following sections explain how to identify and troubleshoot some types of problems.

Performance problems related to running

As a general guideline, if CPU usage is consistently at or above 80 percent, your performance problem is running-related. A running-related problem might be caused by insufficient CPU resources. Or it might be related to one of the following conditions:

- Too many running queries
- Too many compiling queries
- One or more executing queries that use a suboptimal query plan

If you find a running-related performance problem, your goal is to identify the precise problem by using one or more methods. These methods are the most common ways to identify running-related problems:

- Use the [Azure portal](#) to monitor CPU percentage usage.
- Use the following [DMVs](#):
 - The `sys.dm_db_resource_stats` DMV returns CPU, I/O, and memory consumption for an SQL database. One row exists for every 15-second interval, even if there's no activity in the database. Historical data is maintained for one hour.
 - The `sys.resource_stats` DMV returns CPU usage and storage data for Azure SQL Database. The data is collected and aggregated in five-minute intervals.

IMPORTANT

To troubleshoot CPU usage problems for T-SQL queries that use the sys.dm_db_resource_stats and sys.resource_stats DMVs, see [Identify CPU performance issues](#).

Queries that have PSP problems

A parameter sensitive plan (PSP) problem happens when the query optimizer generates a query execution plan that's optimal only for a specific parameter value (or set of values) and the cached plan is then not optimal for parameter values that are used in consecutive executions. Plans that aren't optimal can then cause query performance problems and degrade overall workload throughput.

For more information on parameter sniffing and query processing, see the [Query-processing architecture guide](#).

Several workarounds can mitigate PSP problems. Each workaround has associated tradeoffs and drawbacks:

- Use the [RECOMPILE](#) query hint at each query execution. This workaround trades compilation time and increased CPU for better plan quality. The `RECOMPILE` option is often not possible for workloads that require a high throughput.
- Use the [OPTION \(OPTIMIZE FOR...\)](#) query hint to override the actual parameter value with a typical parameter value that produces a plan that's good enough for most parameter value possibilities. This option requires a good understanding of optimal parameter values and associated plan characteristics.
- Use the [OPTION \(OPTIMIZE FOR UNKNOWN\)](#) query hint to override the actual parameter value and instead use the density vector average. You can also do this by capturing the incoming parameter values in local variables and then using the local variables within the predicates instead of using the parameters themselves. For this fix, the average density must be *good enough*.
- Disable parameter sniffing entirely by using the [DISABLE_PARAMETER_SNIFFING](#) query hint.
- Use the [KEEPFIXEDPLAN](#) query hint to prevent recompilations in cache. This workaround assumes that the good-enough common plan is the one in cache already. You can also disable automatic statistics updates to reduce the chances that the good plan will be evicted and a new bad plan will be compiled.
- Force the plan by explicitly using the [USE PLAN](#) query hint by rewriting the query and adding the hint in the query text. Or set a specific plan by using Query Store or by enabling [automatic tuning](#).
- Replace the single procedure with a nested set of procedures that can each be used based on conditional logic and the associated parameter values.
- Create dynamic string execution alternatives to a static procedure definition.

For more information about resolving PSP problems, see these blog posts:

- [I smell a parameter](#)
- [Conor vs. dynamic SQL vs. procedures vs. plan quality for parameterized queries](#)
- [SQL query optimization techniques in SQL Server: Parameter sniffing](#)

Compile activity caused by improper parameterization

When a query has literals, either the database engine automatically parameterizes the statement or a user explicitly parameterizes the statement to reduce the number of compilations. A high number of compilations for a query using the same pattern but different literal values can result in high CPU usage. Similarly, if you only partially parameterize a query that continues to have literals, the database engine doesn't parameterize the query further.

Here's an example of a partially parameterized query:

```
SELECT *
FROM t1 JOIN t2 ON t1.c1 = t2.c1
WHERE t1.c1 = @p1 AND t2.c2 = '961C3970-0E54-4E8E-82B6-5545BE897F8F'
```

In this example, `t1.c1` takes `@p1`, but `t2.c2` continues to take GUID as literal. In this case, if you change the value for `c2`, the query is treated as a different query, and a new compilation will happen. To reduce compilations in this example, you would also parameterize the GUID.

The following query shows the count of queries by query hash to determine whether a query is properly parameterized:

```
SELECT TOP 10
    q.query_hash
    , count (distinct p.query_id) AS number_of_distinct_query_ids
    , min(qt.query_sql_text) AS sampled_query_text
FROM sys.query_store_query_text AS qt
JOIN sys.query_store_query AS q
    ON qt.query_text_id = q.query_text_id
JOIN sys.query_store_plan AS p
    ON q.query_id = p.query_id
JOIN sys.query_store_runtime_stats AS rs
    ON rs.plan_id = p.plan_id
JOIN sys.query_store_runtime_stats_interval AS rsi
    ON rsi.runtime_stats_interval_id = rs.runtime_stats_interval_id
WHERE
    rsi.start_time >= DATEADD(hour, -2, GETUTCDATE())
    AND query_parameterization_type_desc IN ('User', 'None')
GROUP BY q.query_hash
ORDER BY count (distinct p.query_id) DESC
```

Factors that affect query plan changes

A query execution plan recompilation might result in a generated query plan that differs from the original cached plan. An existing original plan might be automatically recompiled for various reasons:

- Changes in the schema are referenced by the query.
- Data changes to the tables are referenced by the query.
- Query context options were changed.

A compiled plan might be ejected from the cache for various reasons, such as:

- Instance restarts.
- Database-scoped configuration changes.
- Memory pressure.
- Explicit requests to clear the cache.

If you use a RECOMPILE hint, a plan won't be cached.

A recompilation (or fresh compilation after cache eviction) can still result in the generation of a query execution plan that's identical to the original. When the plan changes from the prior or original plan, these explanations are likely:

- **Changed physical design:** For example, newly created indexes more effectively cover the requirements of a query. The new indexes might be used on a new compilation if the query optimizer decides that using that new index is more optimal than using the data structure that was originally selected for the first version of the query execution. Any physical changes to the referenced objects might result in a new plan choice at compile time.
- **Server resource differences:** When a plan in one system differs from the plan in another system, resource availability, such as the number of available processors, can influence which plan gets generated. For example, if one system has more processors, a parallel plan might be chosen.
- **Different statistics:** The statistics associated with the referenced objects might have changed or might be

materially different from the original system's statistics. If the statistics change and a recompilation happens, the query optimizer uses the statistics starting from when they changed. The revised statistics' data distributions and frequencies might differ from those of the original compilation. These changes are used to create cardinality estimates. (*Cardinality estimates* are the number of rows that are expected to flow through the logical query tree.) Changes to cardinality estimates might lead you to choose different physical operators and associated orders of operations. Even minor changes to statistics can result in a changed query execution plan.

- **Changed database compatibility level or cardinality estimator version:** Changes to the database compatibility level can enable new strategies and features that might result in a different query execution plan. Beyond the database compatibility level, a disabled or enabled trace flag 4199 or a changed state of the database-scoped configuration QUERY_OPTIMIZER_HOTFIXES can also influence query execution plan choices at compile time. Trace flags 9481 (force legacy CE) and 2312 (force default CE) also affect the plan.

Resolve problem queries or provide more resources

After you identify the problem, you can either tune the problem queries or upgrade the compute size or service tier to increase the capacity of your SQL database to absorb the CPU requirements.

For more information, see [Scale single database resources in Azure SQL Database](#) and [Scale elastic pool resources in Azure SQL Database](#). For information about scaling a managed instance, see [Service-tier resource limits](#).

Performance problems caused by increased workload volume

An increase in application traffic and workload volume can cause increased CPU usage. But you must be careful to properly diagnose this problem. When you see a high-CPU problem, answer these questions to determine whether the increase is caused by changes to the workload volume:

- Are the queries from the application the cause of the high-CPU problem?
- For the top CPU-consuming queries that you can identify:
 - Were multiple execution plans associated with the same query? If so, why?
 - For queries with the same execution plan, were the execution times consistent? Did the execution count increase? If so, the workload increase is likely causing performance problems.

In summary, if the query execution plan didn't execute differently but CPU usage increased along with execution count, the performance problem is likely related to a workload increase.

It's not always easy to identify a workload volume change that's driving a CPU problem. Consider these factors:

- **Changed resource usage:** For example, consider a scenario where CPU usage increased to 80 percent for an extended period of time. CPU usage alone doesn't mean the workload volume changed. Regressions in the query execution plan and changes in data distribution can also contribute to more resource usage even though the application executes the same workload.
- **The appearance of a new query:** An application might drive a new set of queries at different times.
- **An increase or decrease in the number of requests:** This scenario is the most obvious measure of a workload. The number of queries doesn't always correspond to more resource utilization. However, this metric is still a significant signal, assuming other factors are unchanged.

Waiting-related performance problems

If you're sure that your performance problem isn't related to high CPU usage or to running, your problem is related to waiting. Namely, your CPU resources aren't being used efficiently because the CPU is waiting on some other resource. In this case, identify what your CPU resources are waiting on.

These methods are commonly used to show the top categories of wait types:

- Use [Query Store](#) to find wait statistics for each query over time. In Query Store, wait types are combined into wait categories. You can find the mapping of wait categories to wait types in [sys.query_store_wait_stats](#).
- Use [sys.dm_db_wait_stats](#) to return information about all the waits encountered by threads that executed during operation. You can use this aggregated view to diagnose performance problems with Azure SQL Database and also with specific queries and batches.
- Use [sys.dm_os_waiting_tasks](#) to return information about the queue of tasks that are waiting on some resource.

In high-CPU scenarios, Query Store and wait statistics might not reflect CPU usage if:

- High-CPU-consuming queries are still executing.
- The high-CPU-consuming queries were running when a failover happened.

DMVs that track Query Store and wait statistics show results for only successfully completed and timed-out queries. They don't show data for currently executing statements until the statements finish. Use the dynamic management view [sys.dm_exec_requests](#) to track currently executing queries and the associated worker time.

The chart near the beginning of this article shows that the most common waits are:

- Locks (blocking)
- I/O
- Contention related to TempDB
- Memory grant waits

IMPORTANT

For a set of T-SQL queries that use DMVs to troubleshoot waiting-related problems, see:

- [Identify I/O performance issues](#)
- [Identify memory grant waits](#)
- [TigerToolbox waits and latches](#)
- [TigerToolbox usp_whatsup](#)

Improve database performance with more resources

If no actionable items can improve your database performance, you can change the amount of resources available in Azure SQL Database. Assign more resources by changing the [DTU service tier](#) of a single database. Or increase the eDTUs of an elastic pool at any time. Alternatively, if you're using the [vCore-based purchasing model](#), either change the service tier or increase the resources allocated to your database.

For single databases, you can [change service tiers or compute resources](#) on demand to improve database performance. For multiple databases, consider using [elastic pools](#) to scale resources automatically.

Tune and refactor application or database code

You can optimize the application code for the database, change indexes, force plans, or use hints to manually adapt the database to your workload. For information about manual tuning and rewriting the code, see [Performance tuning guidance](#).

Next steps

- To enable automatic tuning in Azure SQL Database and let the automatic tuning feature fully manage your workload, see [Enable automatic tuning](#).
- To use manual tuning, review [Tuning recommendations in the Azure portal](#). Manually apply the

recommendations that improve performance of your queries.

- Change the resources that are available in your database by changing [Azure SQL Database service tiers](#).

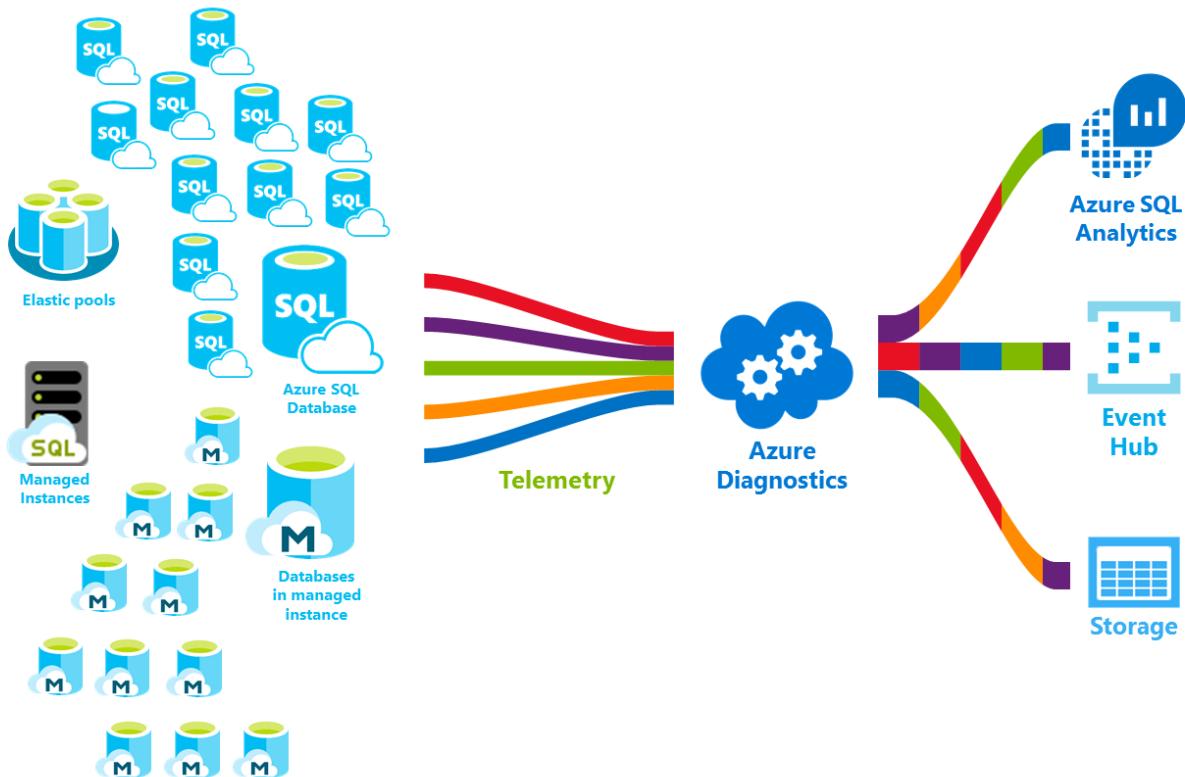
Azure SQL Database metrics and diagnostics logging

2/24/2020 • 24 minutes to read • [Edit Online](#)

In this article, you will learn how to enable and configure logging of diagnostics telemetry for Azure SQL databases through the Azure portal, PowerShell, Azure CLI, the REST API, and Azure Resource Manager template. Single databases, pooled databases, elastic pools, managed instances, and instance databases can stream metrics and diagnostics logs to one of the following Azure resources:

- **Azure SQL Analytics:** Get intelligent monitoring of your databases that includes performance reports, alerts, and mitigation recommendations
- **Azure Event Hubs:** Integrate database telemetry with your custom monitoring solutions or hot pipelines
- **Azure Storage:** Archive vast amounts of telemetry for a fraction of the price

These diagnostics can be used to gauge resource utilization and query execution statistics for easier performance monitoring.



For more information about the metrics and log categories supported by the various Azure services, see:

- [Overview of metrics in Microsoft Azure](#)
- [Overview of Azure diagnostics logs](#)

Enable logging of diagnostics telemetry

You can enable and manage metrics and diagnostics telemetry logging by using one of the following methods:

- Azure portal
- PowerShell
- Azure CLI
- Azure Monitor REST API

- Azure Resource Manager template

When you enable metrics and diagnostics logging, you need to specify the Azure resource destination for collecting the diagnostics telemetry. Available options include:

- [Azure SQL Analytics](#)
- [Azure Event Hubs](#)
- [Azure Storage](#)

You can provision a new Azure resource or select an existing resource. After you choose a resource by using the **Diagnostic settings** option, specify which data to collect.

Supported diagnostic logging for Azure SQL databases

You can set up Azure SQL databases to collect the following diagnostics telemetry:

MONITORING TELEMETRY FOR DATABASES	SINGLE DATABASE AND POOLED DATABASE SUPPORT	MANAGED INSTANCE DATABASE SUPPORT
Basic metrics: Contains DTU/CPU percentage, DTU/CPU limit, physical data read percentage, log write percentage, Successful/Failed/Blocked by firewall connections, sessions percentage, workers percentage, storage, storage percentage, and XTP storage percentage.	Yes	No
Instance and App Advanced: Contains tempdb system database data and log file size and tempdb percent log file used.	Yes	No
QueryStoreRuntimeStatistics: Contains information about the query runtime statistics such as CPU usage and query duration statistics.	Yes	Yes
QueryStoreWaitStatistics: Contains information about the query wait statistics (what your queries waited on) such as CPU, LOG, and LOCKING.	Yes	Yes
Errors: Contains information about SQL errors on a database.	Yes	Yes
DatabaseWaitStatistics: Contains information about how much time a database spent waiting on different wait types.	Yes	No
Timeouts: Contains information about timeouts on a database.	Yes	No
Blocks: Contains information about blocking events on a database.	Yes	No

MONITORING TELEMETRY FOR DATABASES	SINGLE DATABASE AND POOLED DATABASE SUPPORT	MANAGED INSTANCE DATABASE SUPPORT
Deadlocks : Contains information about deadlock events on a database.	Yes	No
AutomaticTuning : Contains information about automatic tuning recommendations for a database.	Yes	No
SQLInsights : Contains Intelligent Insights into performance for a database. To learn more, see Intelligent Insights .	Yes	Yes

IMPORTANT

Elastic pools and managed instances have their own separate diagnostics telemetry from databases they contain. This is important to note as diagnostics telemetry is configured separately for each of these resources.

To enable audit log streaming, see [Set up auditing for your database](#) and [auditing logs in Azure Monitor logs and Azure Event Hubs](#).

Diagnostic settings cannot be configured for the **system databases**, such as master, msdb, model, resource and tempdb databases.

Configure streaming of diagnostic telemetry

You can use the **Diagnostics settings** menu in the Azure portal to enable and configure streaming of diagnostics telemetry. Additionally, you can use PowerShell, the Azure CLI, the [REST API](#), and [Resource Manager templates](#) to configure streaming of diagnostic telemetry. You can set the following destinations to stream the diagnostics telemetry: Azure Storage, Azure Event Hubs, and Azure Monitor logs.

IMPORTANT

Logging of diagnostic telemetry is not enabled by default.

- [Azure portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Elastic pools

You can set up an elastic pool resource to collect the following diagnostics telemetry:

RESOURCE	MONITORING TELEMETRY
Elastic pool	Basic metrics contains eDTU/CPU percentage, eDTU/CPU limit, physical data read percentage, log write percentage, sessions percentage, workers percentage, storage, storage percentage, storage limit, and XTP storage percentage.

To configure streaming of diagnostics telemetry for elastic pools and pooled databases, you need to separately configure each separately:

- Enable streaming of diagnostics telemetry for an elastic pool

- Enable streaming of diagnostics telemetry for each database in elastic pool

The elastic pool container has its own telemetry separate from each individual pooled database's telemetry.

To enable streaming of diagnostics telemetry for an elastic pool resource, follow these steps:

1. Go to the **elastic pool** resource in Azure portal.
2. Select **Diagnostics settings**.
3. Select **Turn on diagnostics** if no previous settings exist, or select **Edit setting** to edit a previous setting.

4. Enter a setting name for your own reference.
5. Select a destination resource for the streaming diagnostics data: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**.
6. For log analytics, select **Configure** and create a new workspace by selecting **+Create New Workspace**, or select an existing workspace.
7. Select the check box for elastic pool diagnostics telemetry: **Basic metrics**.

Diagnostics settings

[Save](#) [Discard](#)

[Delete](#)

* Name

service



Archive to a storage account

Stream to an event hub

Send to Log Analytics

Subscription

Workload Insight dev/test subscription

Log Analytics Workspace

sqlanalytics356 (westcentralus)

METRIC

Basic

8. Select **Save**.

9. In addition, configure streaming of diagnostics telemetry for each database within the elastic pool you want to monitor by following steps described in the next section.

IMPORTANT

In addition to configuring diagnostics telemetry for an elastic pool, you also need to configure diagnostics telemetry for each database in the elastic pool.

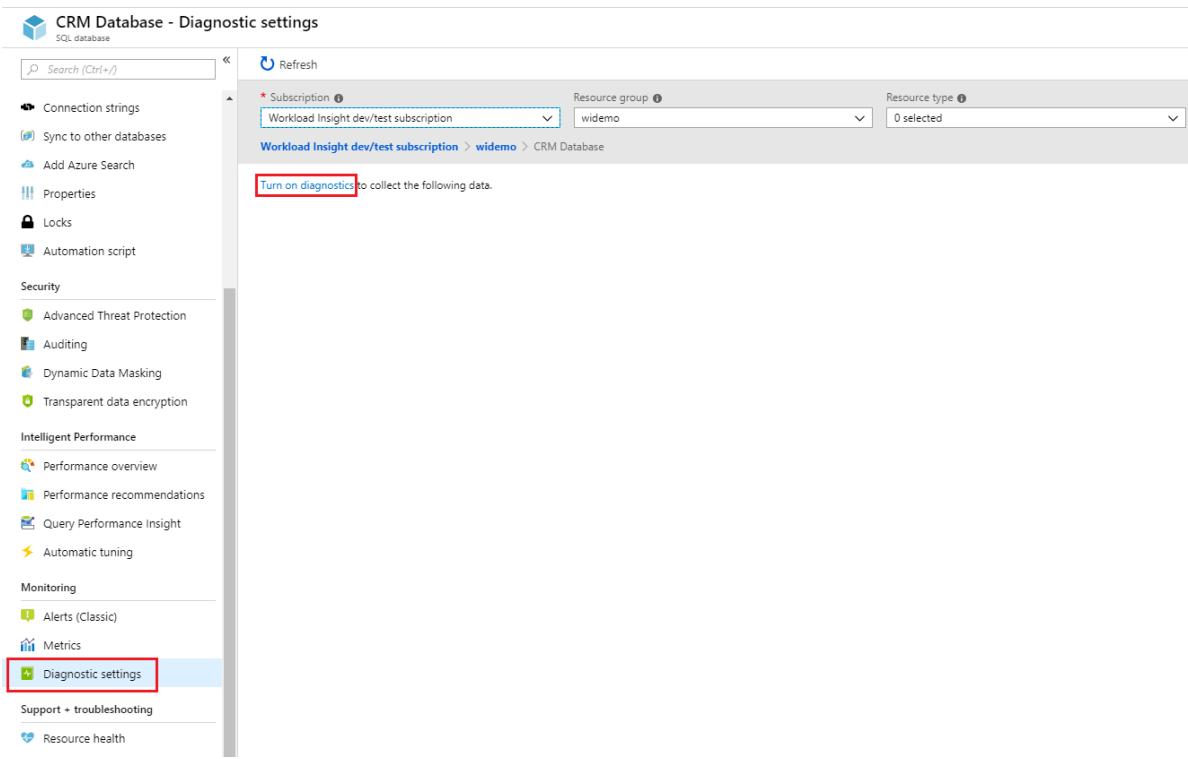
Single or pooled database

You can set up a single or pooled database resource to collect the following diagnostics telemetry:

RESOURCE	MONITORING TELEMETRY
Single or pooled database	Basic metrics contains DTU percentage, DTU used, DTU limit, CPU percentage, physical data read percentage, log write percentage, Successful/Failed/Blocked by firewall connections, sessions percentage, workers percentage, storage, storage percentage, XTP storage percentage, and deadlocks.

To enable streaming of diagnostics telemetry for a single or a pooled database, follow these steps:

1. Go to Azure **SQL database** resource.
2. Select **Diagnostics settings**.
3. Select **Turn on diagnostics** if no previous settings exist, or select **Edit setting** to edit a previous setting.
You can create up to three parallel connections to stream diagnostics telemetry.
4. Select **Add diagnostic setting** to configure parallel streaming of diagnostics data to multiple resources.



5. Enter a setting name for your own reference.
6. Select a destination resource for the streaming diagnostics data: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**.
7. For the standard, event-based monitoring experience, select the following check boxes for database diagnostics log telemetry: **SQLInsights**, **AutomaticTuning**, **QueryStoreRuntimeStatistics**, **QueryStoreWaitStatistics**, **Errors**, **DatabaseWaitStatistics**, **Timeouts**, **Blocks**, and **Deadlocks**.
8. For an advanced, one-minute-based monitoring experience, select the check box for **Basic** metrics.

Diagnostics settings

X

Save Discard Delete

* Name

service



Archive to a storage account

Stream to an event hub

Send to Log Analytics

Subscription

Workload Insight dev/test subscription

Log Analytics Workspace

sqlanalytics356 (westcentralus)

LOG

SQLInsights

AutomaticTuning

QueryStoreRuntimeStatistics

QueryStoreWaitStatistics

Errors

DatabaseWaitStatistics

Timeouts

Blocks

Deadlocks

METRIC

Basic

9. Select **Save**.

10. Repeat these steps for each database you want to monitor.

TIP

Repeat these steps for each single and pooled database you want to monitor.

Managed instance

You can set up a managed instance resource to collect the following diagnostics telemetry:

RESOURCE	MONITORING TELEMETRY
Managed instance	ResourceUsageStats contains vCores count, average CPU percentage, IO requests, bytes read/written, reserved storage space, and used storage space.

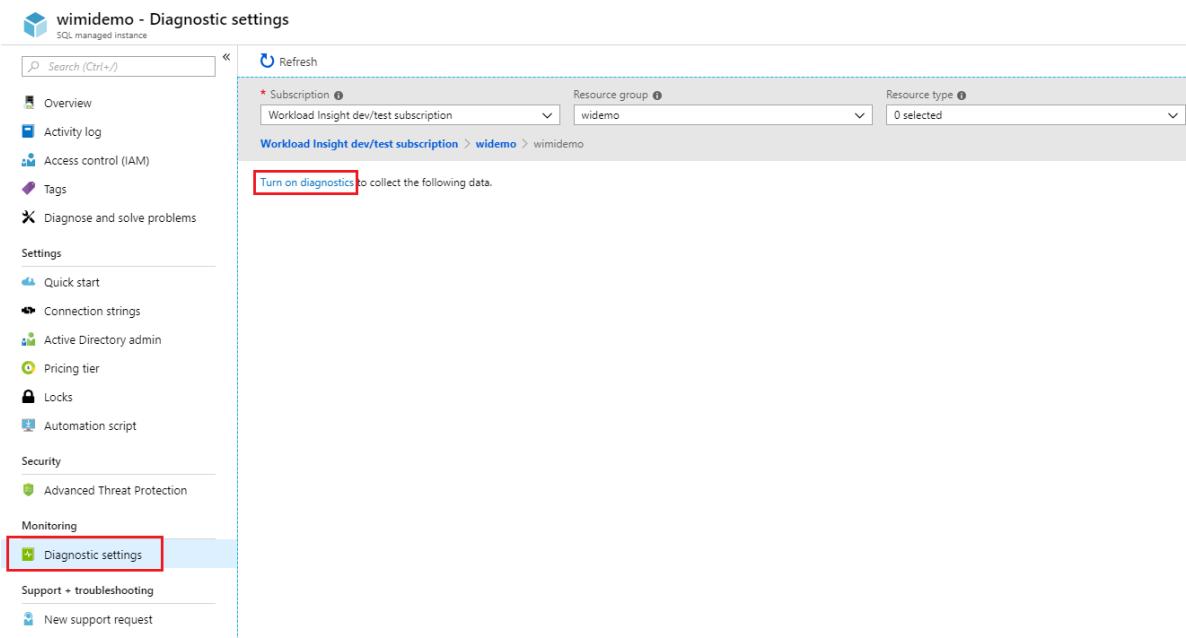
To configure streaming of diagnostics telemetry for managed instance and instance databases, you will need to separately configure each:

- Enable streaming of diagnostics telemetry for managed instance
- Enable streaming of diagnostics telemetry for each instance database

The managed instance container has its own telemetry separate from each instance database's telemetry.

To enable streaming of diagnostics telemetry for a managed instance resource, follow these steps:

1. Go to the **managed instance** resource in Azure portal.
2. Select **Diagnostics settings**.
3. Select **Turn on diagnostics** if no previous settings exist, or select **Edit setting** to edit a previous setting.



4. Enter a setting name for your own reference.
5. Select a destination resource for the streaming diagnostics data: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**.
6. For log analytics, select **Configure** and create a new workspace by selecting **+Create New Workspace**, or use an existing workspace.
7. Select the check box for instance diagnostics telemetry: **ResourceUsageStats**.

Diagnostics settings

[Save](#) [Discard](#) [Delete](#)

* Name

service

X

Archive to a storage account

Stream to an event hub

Send to Log Analytics

Subscription

Workload Insight dev/test subscription

Log Analytics Workspace

sqianalytics356 (westcentralus)

LOG

ResourceUsageStats

8. Select **Save**.

9. In addition, configure streaming of diagnostics telemetry for each instance database within the managed instance you want to monitor by following the steps described in the next section.

IMPORTANT

In addition to configuring diagnostics telemetry for a managed instance, you also need to configure diagnostics telemetry for each instance database.

Instance database

You can set up an instance database resource to collect the following diagnostics telemetry:

RESOURCE	MONITORING TELEMETRY
Instance database	ResourceUsageStats contains vCores count, average CPU percentage, IO requests, bytes read/written, reserved storage space, and used storage space.

To enable streaming of diagnostics telemetry for an instance database, follow these steps:

1. Go to **instance database** resource within managed instance.
2. Select **Diagnostics settings**.
3. Select **Turn on diagnostics** if no previous settings exist, or select **Edit setting** to edit a previous setting.
 - You can create up to three (3) parallel connections to stream diagnostics telemetry.
 - Select **+Add diagnostic setting** to configure parallel streaming of diagnostics data to multiple resources.

The screenshot shows the 'CRM Database - Diagnostic settings' page in the Azure portal. On the left, there's a navigation menu with options like Overview, Activity log, Diagnose and solve problems, Settings, Security, Monitoring, and Diagnostic settings. The 'Diagnostic settings' option is highlighted with a red box. The main area shows a subscription dropdown set to 'Workload Insight dev/test subscription', a resource group dropdown set to 'widemo', and a resource type dropdown set to '0 selected'. Below these is a button labeled 'Turn on diagnostics'.

4. Enter a setting name for your own reference.
5. Select a destination resource for the streaming diagnostics data: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**.
6. Select the check boxes for database diagnostics telemetry: **SQLInsights**, **QueryStoreRuntimeStatistics**, **QueryStoreWaitStatistics**, and **Errors**.

Diagnostics settings

Save Discard Delete

Name: service

Archive to a storage account

Stream to an event hub

Send to Log Analytics

Subscription: Workload Insight dev/test subscription

Log Analytics Workspace: sqlanalytics356 (westcentralus)

LOG

SQLInsights

QueryStoreRuntimeStatistics

QueryStoreWaitStatistics

Errors

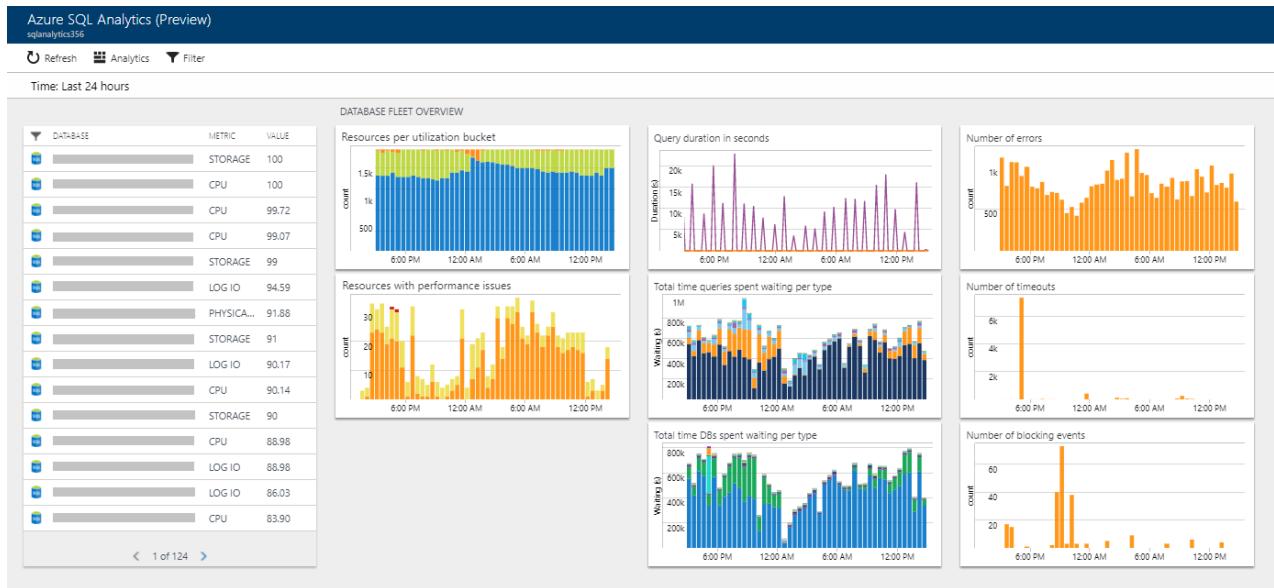
7. Select **Save**.
8. Repeat these steps for each instance database you want to monitor.

TIP

Repeat these steps for each instance database you want to monitor.

Stream diagnostic telemetry into SQL Analytics

Azure SQL Analytics is a cloud solution that monitors the performance of single databases, elastic pools and pooled databases, and managed instances and instance databases at scale and across multiple subscriptions. It can help you collect and visualize Azure SQL Database performance metrics, and it has built-in intelligence for performance troubleshooting.



SQL Database metrics and diagnostics logs can be streamed into Azure SQL Analytics by using the built-in **Send to Log Analytics** option in the diagnostics settings tab in the Azure portal. You can also enable log analytics by using diagnostics settings via PowerShell cmdlets, the Azure CLI, or the Azure Monitor REST API.

Installation overview

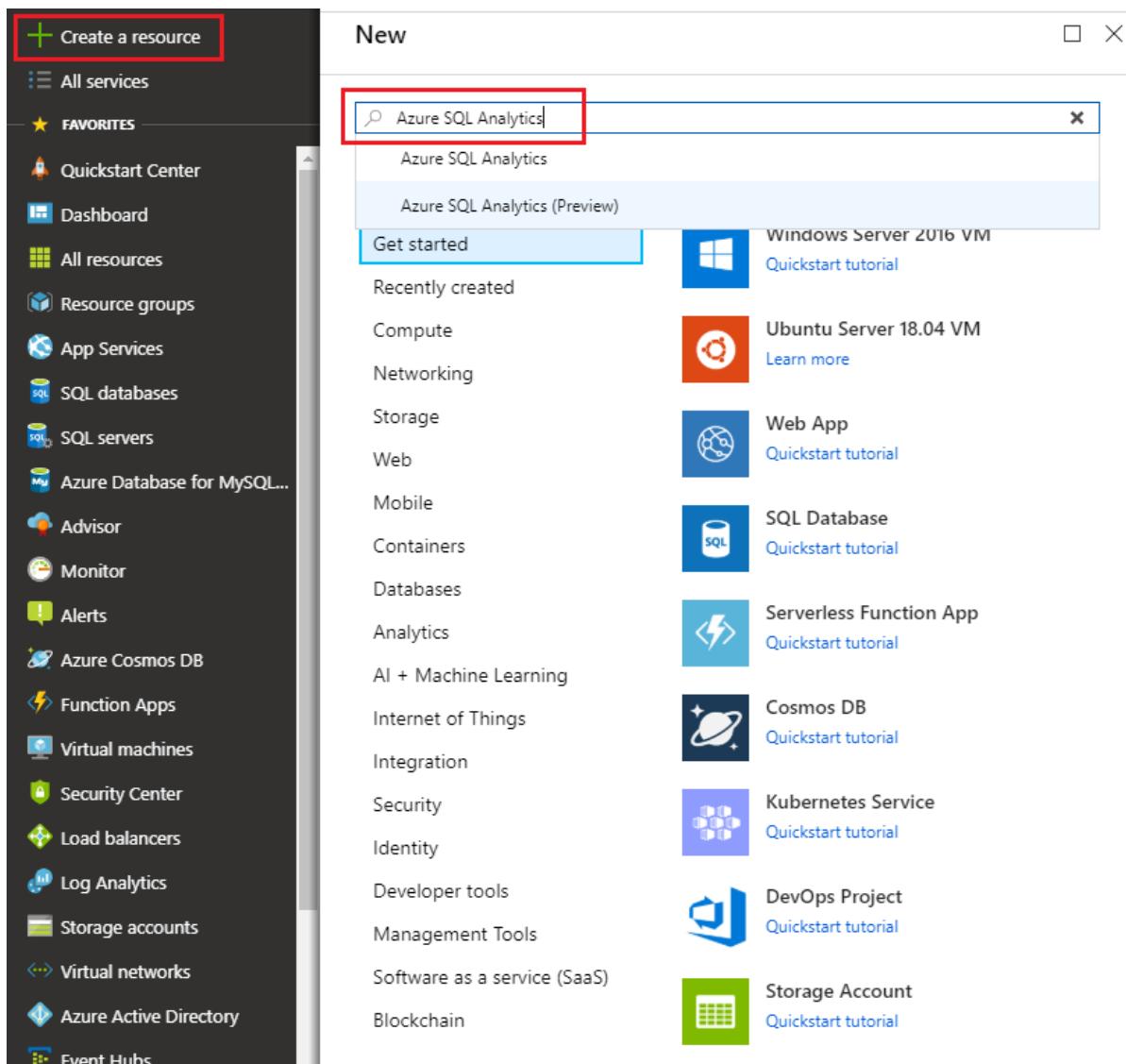
You can monitor a collection of Azure SQL databases with Azure SQL Analytics by performing the following steps:

1. Create an Azure SQL Analytics solution from the Azure Marketplace.
2. Create a monitoring workspace in the solution.
3. Configure databases to stream diagnostics telemetry into the workspace.

If you're using elastic pools or managed instances, you also need to configure diagnostics telemetry streaming from these resources.

Create an Azure SQL Analytics resource

1. Search for Azure SQL Analytics in Azure Marketplace and select it.



2. Select **Create** on the solution's overview screen.
3. Fill in the Azure SQL Analytics form with the additional information that is required: workspace name, subscription, resource group, location, and pricing tier.

The cost of your workspace depends on the pricing tier and what solutions you use. Learn more about [Log Analytics pricing](#).

Pricing Tier
<input checked="" type="radio"/> Free
<input type="radio"/> Per Node (OMS)
<input type="radio"/> Per GB (Standalone)

4. Select **OK** to confirm, and then select **Create**.

Configure databases to record metrics and diagnostics logs

The easiest way to configure where databases record metrics is by using the Azure portal. Go to your database resource in the Azure portal and select **Diagnostics settings**.

Use Azure SQL Analytics for monitoring and alerting

You can use SQL Analytics as a hierarchical dashboard to view your SQL database resources.

- To learn how to use Azure SQL Analytics, see [Monitor SQL Database by using SQL Analytics](#).
- To learn how to set up alerts for in SQL Analytics, see [Creating alerts for database, elastic pools, and managed instances](#).

Stream diagnostic telemetry into Event Hubs

You can stream SQL Database metrics and diagnostics logs into Event Hubs by using the built-in **Stream to an event hub** option in the Azure portal. You also can enable the Service Bus rule ID by using diagnostics settings via PowerShell cmdlets, the Azure CLI, or the Azure Monitor REST API.

What to do with metrics and diagnostics logs in Event Hubs

After the selected data is streamed into Event Hubs, you're one step closer to enabling advanced monitoring scenarios. Event Hubs acts as the front door for an event pipeline. After data is collected into an event hub, it can be transformed and stored by using a real-time analytics provider or a storage adapter. Event Hubs decouples the production of a stream of events from the consumption of those events. In this way, event consumers can access the events on their own schedule. For more information on Event Hubs, see:

- [What are Azure Event Hubs?](#)
- [Get started with Event Hubs](#)

You can use streamed metrics in Event Hubs to:

- **View service health by streaming hot-path data to Power BI**

By using Event Hubs, Stream Analytics, and Power BI, you can easily transform your metrics and diagnostics data into near real-time insights on your Azure services. For an overview of how to set up an event hub, process data with Stream Analytics, and use Power BI as an output, see [Stream Analytics and Power BI](#).

- **Stream logs to third-party logging and telemetry streams**

By using Event Hubs streaming, you can get your metrics and diagnostics logs into various third-party monitoring and log analytics solutions.

- **Build a custom telemetry and logging platform**

Do you already have a custom-built telemetry platform or are considering building one? The highly scalable publish-subscribe nature of Event Hubs allows you to flexibly ingest diagnostics logs. See [Dan Rosanova's guide to using Event Hubs in a global-scale telemetry platform](#).

Stream diagnostic telemetry into Azure Storage

You can store metrics and diagnostics logs in Azure Storage by using the built-in **Archive to a storage account** option in the Azure portal. You can also enable Storage by using diagnostics settings via PowerShell cmdlets, the Azure CLI, or the Azure Monitor REST API.

Schema of metrics and diagnostics logs in the storage account

After you set up metrics and diagnostics logs collection, a storage container is created in the storage account you selected when the first rows of data are available. The structure of the blobs is:

```
insights-{metrics|logs}-{category name}/resourceId=/SUBSCRIPTIONS/{subscription ID}/ RESOURCEGROUPS/{resource group name}/PROVIDERS/Microsoft.SQL/servers/{resource_server}/ databases/{database_name}/y={four-digit numeric year}/m={two-digit numeric month}/d={two-digit numeric day}/h={two-digit 24-hour clock hour}/m=0/PT1H.json
```

Or, more simply:

```
insights-{metrics|logs}-{category name}/resourceId=/resource Id}/y={four-digit numeric year}/m={two-digit numeric month}/d={two-digit numeric day}/h={two-digit 24-hour clock hour}/m=00/PT1H.json
```

For example, a blob name for Basic metrics might be:

```
insights-metrics-minute/resourceId=/SUBSCRIPTIONS/s1id1234-5679-0123-4567-890123456789/RESOURCEGROUPS/TESTRESOURCEGROUP/PROVIDERS/MICROSOFT.SQL/servers/Server1/databases/database1/y=2016/m=08/d=22/h=18/m=00/PT1H.json
```

A blob name for storing data from an elastic pool looks like:

```
insights-{metrics|logs}-{category name}/resourceId=/SUBSCRIPTIONS/{subscription ID}/ RESOURCEGROUPS/{resource group name}/PROVIDERS/Microsoft.SQL/servers/{resource_server}/ elasticPools/{elastic_pool_name}/y={four-digit numeric year}/m={two-digit numeric month}/d={two-digit numeric day}/h={two-digit 24-hour clock hour}/m=00/PT1H.json
```

Data retention policy and pricing

If you select Event Hubs or a Storage account, you can specify a retention policy. This policy deletes data that is older than a selected time period. If you specify Log Analytics, the retention policy depends on the selected pricing tier. In this case, the provided free units of data ingestion can enable free monitoring of several databases each month. Any consumption of diagnostics telemetry in excess of the free units might incur costs.

IMPORTANT

Active databases with heavier workloads ingest more data than idle databases. For more information, see [Log analytics pricing](#).

If you are using Azure SQL Analytics, you can monitor your data ingestion consumption by selecting **OMS Workspace** on the navigation menu of Azure SQL Analytics, and then selecting **Usage** and **Estimated Costs**.

Metrics and logs available

Monitoring telemetry available for single databases, pooled databases, elastic pools, managed instance, and instance databases is documented in this section of the article. Collected monitoring telemetry inside SQL Analytics can be used for your own custom analysis and application development using [Azure Monitor log queries](#) language.

Basic metrics

Refer to the following tables for details about Basic metrics by resource.

NOTE

Basic metrics option was formerly known as All metrics. The change made was to the naming only and there was no change to the metrics monitored. This change was initiated to allow for introduction of additional metric categories in the future.

Basic metrics for elastic pools

RESOURCE	METRICS
Elastic pool	eDTU percentage, eDTU used, eDTU limit, CPU percentage, physical data read percentage, log write percentage, sessions percentage, workers percentage, storage, storage percentage, storage limit, XTP storage percentage

Basic metrics for single and pooled databases

RESOURCE	METRICS
Single and pooled database	DTU percentage, DTU used, DTU limit, CPU percentage, physical data read percentage, log write percentage, Successful/Failed/Blocked by firewall connections, sessions percentage, workers percentage, storage, storage percentage, XTP storage percentage, and deadlocks

Advanced metrics

Refer to the following table for details about advanced metrics.

METRIC	METRIC DISPLAY NAME	DESCRIPTION
tempdb_data_size	Tempdb Data File Size Kilobytes	Tempdb Data File Size Kilobytes. Not applicable to data warehouses. This metric will be available for databases using the vCore purchasing model with 2 vCores and higher, or 200 DTU and higher for DTU-based purchasing models. This metric is not currently available for Hyperscale databases.
tempdb_log_size	Tempdb Log File Size Kilobytes	Tempdb Log File Size Kilobytes. Not applicable to data warehouses. This metric will be available for databases using the vCore purchasing model with 2 vCores and higher, or 200 DTU and higher for DTU-based purchasing models. This metric is not currently available for Hyperscale databases.
tempdb_log_used_percent	Tempdb Percent Log Used	Tempdb Percent Log Used. Not applicable to data warehouses. This metric will be available for databases using the vCore purchasing model with 2 vCores and higher, or 200 DTU and higher for DTU-based purchasing models. This metric is not currently available for Hyperscale databases.

Basic logs

Details of telemetry available for all logs are documented in the following tables. See [supported diagnostic logging](#) to understand which logs are supported for a particular database flavor - Azure SQL single, pooled, or instance database.

Resource usage stats for managed instances

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: ResourceUsageStats
Resource	Name of the resource
ResourceType	Name of the resource type. Always: MANAGEDINSTANCES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the managed instance
ResourceId	Resource URI
SKU_s	Managed instance product SKU
virtual_core_count_s	Number of vCores available
avg_cpu_percent_s	Average CPU percentage
reserved_storage_mb_s	Reserved storage capacity on the managed instance
storage_space_used_mb_s	Used storage on the managed instance
io_requests_s	IOPS count
io_bytes_read_s	IOPS bytes read
io_bytes_written_s	IOPS bytes written

Query Store runtime statistics

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics

PROPERTY	DESCRIPTION
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: QueryStoreRuntimeStatistics
OperationName	Name of the operation. Always: QueryStoreRuntimeStatisticsEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
query_hash_s	Query hash
query_plan_hash_s	Query plan hash
statement_sql_handle_s	Statement sql handle
interval_start_time_d	Start datetimeoffset of the interval in number of ticks from 1900-1-1
interval_end_time_d	End datetimeoffset of the interval in number of ticks from 1900-1-1
logical_io_writes_d	Total number of logical IO writes
max_logical_io_writes_d	Max number of logical IO writes per execution
physical_io_reads_d	Total number of physical IO reads
max_physical_io_reads_d	Max number of logical IO reads per execution
logical_io_reads_d	Total number of logical IO reads
max_logical_io_reads_d	Max number of logical IO reads per execution
execution_type_d	Execution type
count_executions_d	Number of executions of the query

PROPERTY	DESCRIPTION
cpu_time_d	Total CPU time consumed by the query in microseconds
max_cpu_time_d	Max CPU time consumer by a single execution in microseconds
dop_d	Sum of degrees of parallelism
max_dop_d	Max degree of parallelism used for single execution
rowcount_d	Total number of rows returned
max_rowcount_d	Max number of rows returned in single execution
query_max_used_memory_d	Total amount of memory used in KB
max_query_max_used_memory_d	Max amount of memory used by a single execution in KB
duration_d	Total execution time in microseconds
max_duration_d	Max execution time of a single execution
num_physical_io_reads_d	Total number of physical reads
max_num_physical_io_reads_d	Max number of physical reads per execution
log_bytes_used_d	Total amount of log bytes used
max_log_bytes_used_d	Max amount of log bytes used per execution
query_id_d	ID of the query in Query Store
plan_id_d	ID of the plan in Query Store

Learn more about [Query Store runtime statistics data](#).

Query Store wait statistics

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: QueryStoreWaitStatistics

PROPERTY	DESCRIPTION
OperationName	Name of the operation. Always: QueryStoreWaitStatisticsEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
wait_category_s	Category of the wait
is_parameterizable_s	Is the query parameterizable
statement_type_s	Type of the statement
statement_key_hash_s	Statement key hash
exec_type_d	Type of execution
total_query_wait_time_ms_d	Total wait time of the query on the specific wait category
max_query_wait_time_ms_d	Max wait time of the query in individual execution on the specific wait category
query_param_type_d	0
query_hash_s	Query hash in Query Store
query_plan_hash_s	Query plan hash in Query Store
statement_sql_handle_s	Statement handle in Query Store
interval_start_time_d	Start datetimeoffset of the interval in number of ticks from 1900-1-1
interval_end_time_d	End datetimeoffset of the interval in number of ticks from 1900-1-1
count_executions_d	Count of executions of the query
query_id_d	ID of the query in Query Store

PROPERTY	DESCRIPTION
plan_id_d	ID of the plan in Query Store

Learn more about [Query Store wait statistics data](#).

Errors dataset

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: Errors
OperationName	Name of the operation. Always: ErrorEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
Message	Error message in plain text
user_defined_b	Is the error user defined bit
error_number_d	Error code
Severity	Severity of the error
state_d	State of the error
query_hash_s	Query hash of the failed query, if available
query_plan_hash_s	Query plan hash of the failed query, if available

Learn more about [SQL Server error messages](#).

Database wait statistics dataset

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: DatabaseWaitStatistics
OperationName	Name of the operation. Always: DatabaseWaitStatisticsEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
wait_type_s	Name of the wait type
start_utc_date_t [UTC]	Measured period start time
end_utc_date_t [UTC]	Measured period end time
delta_max_wait_time_ms_d	Max waited time per execution
delta_signal_wait_time_ms_d	Total signals wait time
delta_wait_time_ms_d	Total wait time in the period
delta_waiting_tasks_count_d	Number of waiting tasks

Learn more about [database wait statistics](#).

Time-outs dataset

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: Timeouts
OperationName	Name of the operation. Always: TimeoutEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
error_state_d	Error state code
query_hash_s	Query hash, if available
query_plan_hash_s	Query plan hash, if available

Blockings dataset

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: Blocks

PROPERTY	DESCRIPTION
OperationName	Name of the operation. Always: BlockEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
lock_mode_s	Lock mode used by the query
resource_owner_type_s	Owner of the lock
blocked_process_filtered_s	Blocked process report XML
duration_d	Duration of the lock in microseconds

Deadlocks dataset

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: Deadlocks
OperationName	Name of the operation. Always: DeadlockEvent
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database

PROPERTY	DESCRIPTION
LogicalServerName_s	Name of the server for the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
deadlock_xml_s	Deadlock report XML

Automatic tuning dataset

PROPERTY	DESCRIPTION
TenantId	Your tenant ID
SourceSystem	Always: Azure
TimeGenerated [UTC]	Time stamp when the log was recorded
Type	Always: AzureDiagnostics
ResourceProvider	Name of the resource provider. Always: MICROSOFT.SQL
Category	Name of the category. Always: AutomaticTuning
Resource	Name of the resource
ResourceType	Name of the resource type. Always: SERVERS/DATABASES
SubscriptionId	Subscription GUID for the database
ResourceGroup	Name of the resource group for the database
LogicalServerName_s	Name of the server for the database
LogicalDatabaseName_s	Name of the database
ElasticPoolName_s	Name of the elastic pool for the database, if any
DatabaseName_s	Name of the database
ResourceId	Resource URI
RecommendationHash_s	Unique hash of Automatic tuning recommendation
OptionName_s	Automatic tuning operation
Schema_s	Database schema
Table_s	Table affected

PROPERTY	DESCRIPTION
IndexName_s	Index name
IndexColumns_s	Column name
IncludedColumns_s	Columns included
EstimatedImpact_s	Estimated impact of Automatic tuning recommendation JSON
Event_s	Type of Automatic tuning event
Timestamp_t	Last updated timestamp

Intelligent Insights dataset

Learn more about the [Intelligent Insights log format](#).

Next steps

To learn how to enable logging and to understand the metrics and log categories supported by the various Azure services, see:

- [Overview of metrics in Microsoft Azure](#)
- [Overview of Azure diagnostics logs](#)

To learn about Event Hubs, read:

- [What is Azure Event Hubs?](#)
- [Get started with Event Hubs](#)

To learn how to set up alerts based on telemetry from log analytics see:

- [Creating alerts for SQL Database and managed instance](#)

Automatic tuning in Azure SQL Database

2/20/2020 • 5 minutes to read • [Edit Online](#)

Azure SQL Database Automatic tuning provides peak performance and stable workloads through continuous performance tuning based on AI and machine learning.

Automatic tuning is a fully managed intelligent performance service that uses built-in intelligence to continuously monitor queries executed on a database, and it automatically improves their performance. This is achieved through dynamically adapting database to the changing workloads and applying tuning recommendations. Automatic tuning learns horizontally from all databases on Azure through AI and it dynamically improves its tuning actions. The longer an Azure SQL Database runs with automatic tuning on, the better it performs.

Azure SQL Database Automatic tuning might be one of the most important features that you can enable to provide stable and peak performing database workloads.

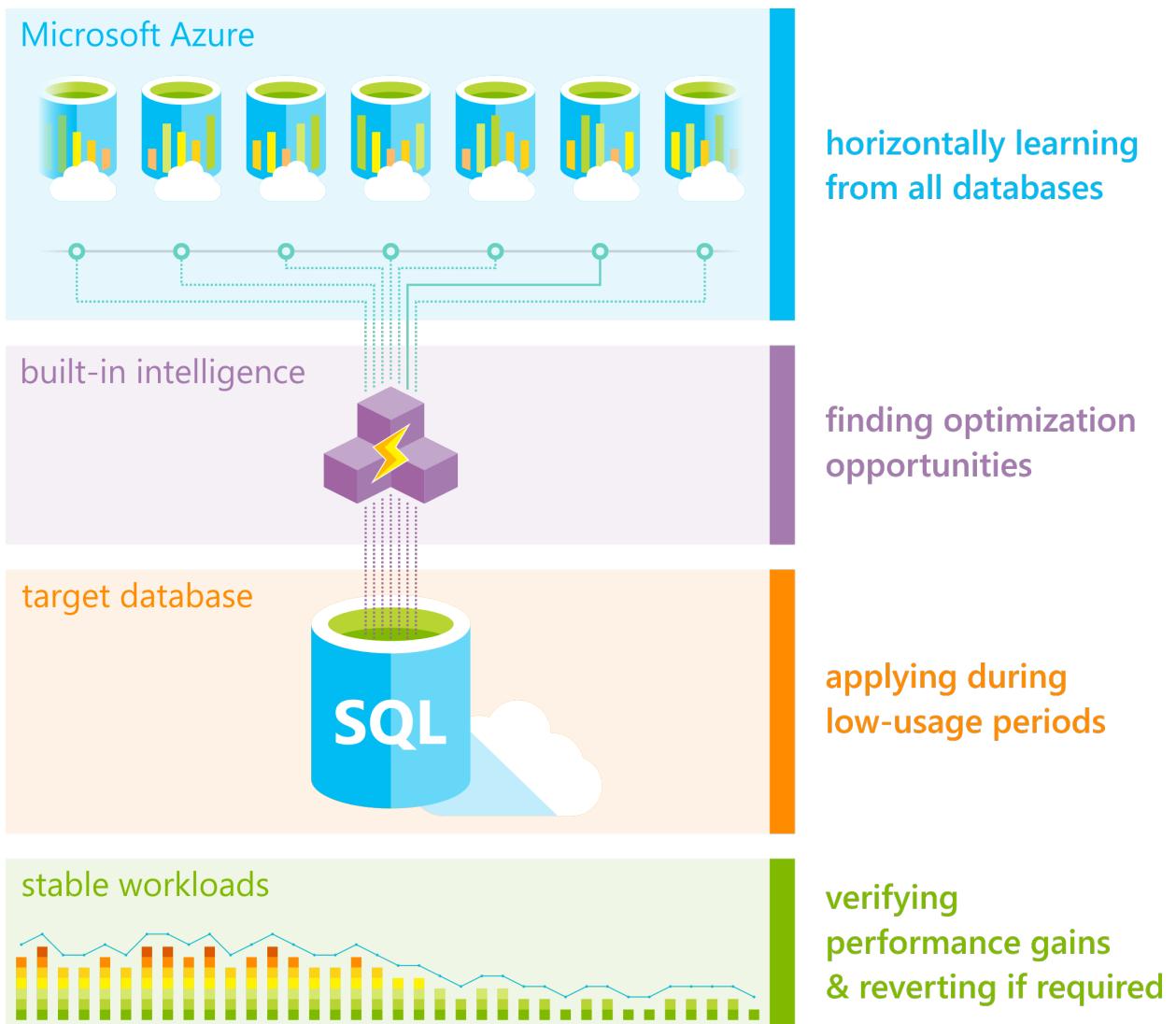
What can Automatic Tuning do for you?

- Automated performance tuning of Azure SQL databases
- Automated verification of performance gains
- Automated rollback and self-correction
- Tuning history
- Tuning action T-SQL scripts for manual deployments
- Proactive workload performance monitoring
- Scale out capability on hundreds of thousands of databases
- Positive impact to DevOps resources and the total cost of ownership

Safe, Reliable, and Proven

Tuning operations applied to Azure SQL databases are fully safe for the performance of your most intense workloads. The system has been designed with care not to interfere with the user workloads. Automated tuning recommendations are applied only at the times of a low utilization. The system can also temporarily disable automatic tuning operations to protect the workload performance. In such case, "Disabled by the system" message will be shown in Azure portal. Automatic tuning regards workloads with the highest resource priority.

Automatic tuning mechanisms are mature and have been perfected on several million databases running on Azure. Automated tuning operations applied are verified automatically to ensure there is a positive improvement to the workload performance. Regressed performance recommendations are dynamically detected and promptly reverted. Through the tuning history recorded, there exists a clear trace of tuning improvements made to each Azure SQL Database.



Azure SQL Database Automatic tuning is sharing its core logic with the SQL Server automatic tuning engine. For additional technical information on the built-in intelligence mechanism, see [SQL Server automatic tuning](#).

Use Automatic tuning

Automatic tuning needs to be enabled on your subscription. To enable automatic tuning using Azure portal, see [Enable automatic tuning](#).

Automatic tuning can operate autonomously through automatically applying tuning recommendations, including automated verification of performance gains.

For more control, automatic application of tuning recommendations can be turned off, and tuning recommendations can be manually applied through Azure portal. It is also possible to use the solution to view automated tuning recommendations only and manually apply them through scripts and tools of your choice.

For an overview of how automatic tuning works and for typical usage scenarios, see the embedded video:

Automatic tuning options

Automatic tuning options available in Azure SQL Database are:

AUTOMATIC TUNING OPTION	SINGLE DATABASE AND POOLED DATABASE SUPPORT	INSTANCE DATABASE SUPPORT
CREATE INDEX - Identifies indexes that may improve performance of your workload, creates indexes, and automatically verifies that performance of queries has improved.	Yes	No
DROP INDEX - Identifies redundant and duplicate indexes daily, except for unique indexes, and indexes that were not used for a long time (>90 days). Please note that this option is not compatible with applications using partition switching and index hints. Dropping unused indexes is not supported for Premium and Business Critical service tiers.	Yes	No
FORCE LAST GOOD PLAN (automatic plan correction) - Identifies SQL queries using execution plan that is slower than the previous good plan, and queries using the last known good plan instead of the regressed plan.	Yes	Yes

Automatic tuning identifies **CREATE INDEX**, **DROP INDEX**, and **FORCE LAST GOOD PLAN** recommendations that can optimize your database performance and shows them in [Azure portal](#), and exposes them through [T-SQL](#) and [REST API](#). To learn more about FORCE LAST GOOD PLAN and configuring automatic tuning options through T-SQL, see [Automatic tuning introduces automatic plan correction](#).

You can either manually apply tuning recommendations using the portal or you can let Automatic tuning autonomously apply tuning recommendations for you. The benefits of letting the system autonomously apply tuning recommendations for you is that it automatically validates there exists a positive gain to the workload performance, and if there is no significant performance improvement detected, it will automatically revert the tuning recommendation. Please note that in case of queries affected by tuning recommendations that are not executed frequently, the validation phase can take up to 72 hrs by design.

In case you are applying tuning recommendations through T-SQL, the automatic performance validation, and reversal mechanisms are not available. Recommendations applied in such way will remain active and shown in the list of tuning recommendations for 24-48 hrs. before the system automatically withdraws them. If you would like to remove a recommendation sooner, you can discard it from Azure portal.

Automatic tuning options can be independently enabled or disabled per database, or they can be configured on SQL Database servers and applied on every database that inherits settings from the server. SQL Database servers can inherit Azure defaults for Automatic tuning settings. Azure defaults at this time are set to FORCE_LAST_GOOD_PLAN is enabled, CREATE_INDEX is enabled, and DROP_INDEX is disabled.

IMPORTANT

As of March, 2020 changes to Azure defaults for automatic tuning will take effect as follows:

- New Azure defaults will be FORCE_LAST_GOOD_PLAN = enabled, CREATE_INDEX = disabled, and DROP_INDEX = disabled.
- Existing servers with no automatic tuning preferences configured will be automatically configured to INHERIT the new Azure defaults. This applies to all customers currently having server settings for automatic tuning in an undefined state.
- New servers created will automatically be configured to INHERIT the new Azure defaults (unlike earlier when automatic tuning configuration was in an undefined state upon new server creation).

Configuring Automatic tuning options on a server and inheriting settings for databases belonging to the parent server is a recommended method for configuring automatic tuning as it simplifies management of automatic tuning options for a large number of databases.

Next steps

- To enable automatic tuning in Azure SQL Database to manage your workload, see [Enable automatic tuning](#).
- To manually review and apply Automatic tuning recommendations, see [Find and apply performance recommendations](#).
- To learn how to use T-SQL to apply and view Automatic tuning recommendations, see [Manage automatic tuning via T-SQL](#).
- To learn about building email notifications for Automatic tuning recommendations, see [Email notifications for automatic tuning](#).
- To learn about built-in intelligence used in Automatic tuning, see [Artificial Intelligence tunes Azure SQL databases](#).
- To learn about how Automatic tuning works in Azure SQL Database and SQL server 2017, see [SQL Server automatic tuning](#).

Intelligent Insights using AI to monitor and troubleshoot database performance (Preview)

2/24/2020 • 10 minutes to read • [Edit Online](#)

Azure SQL Database Intelligent Insights lets you know what is happening with your SQL Database and Managed Instance database performance.

Intelligent Insights uses built-in intelligence to continuously monitor database usage through artificial intelligence and detect disruptive events that cause poor performance. Once detected, a detailed analysis is performed that generates a diagnostics log with an intelligent assessment of the issue. This assessment consists of a root cause analysis of the database performance issue and, where possible, recommendations for performance improvements.

What can Intelligent Insights do for you

Intelligent Insights is a unique capability of Azure built-in intelligence that provides the following value:

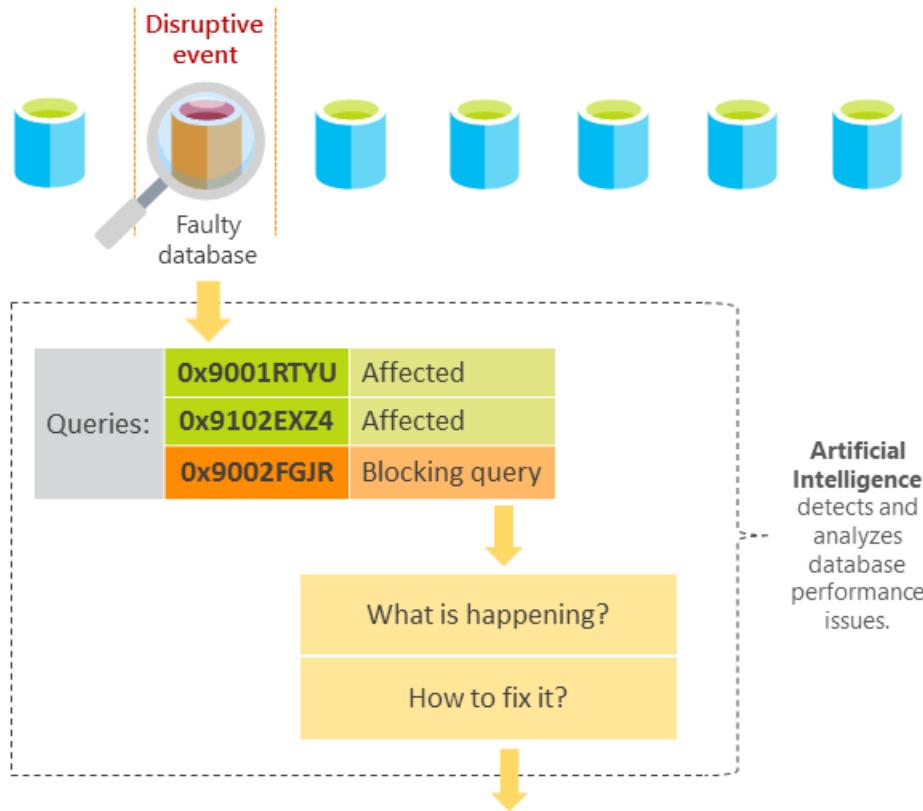
- Proactive monitoring
- Tailored performance insights
- Early detection of database performance degradation
- Root cause analysis of issues detected
- Performance improvement recommendations
- Scale out capability on hundreds of thousands of databases
- Positive impact to DevOps resources and the total cost of ownership

How does Intelligent Insights work

Intelligent Insights analyzes database performance by comparing the database workload from the last hour with the past seven-day baseline workload. Database workload is composed of queries determined to be the most significant to the database performance, such as the most repeated and largest queries. Because each database is unique based on its structure, data, usage, and application, each workload baseline that is generated is specific and unique to an individual instance. Intelligent Insights, independent of the workload baseline, also monitors absolute operational thresholds and detects issues with excessive wait times, critical exceptions, and issues with query parameterizations that might affect performance.

After a performance degradation issue is detected from multiple observed metrics by using artificial intelligence, analysis is performed. A diagnostics log is generated with an intelligent insight on what is happening with your database. Intelligent Insights makes it easy to track the database performance issue from its first appearance until resolution. Each detected issue is tracked through its lifecycle from initial issue detection and verification of performance improvement to its completion.

Azure SQL Databases



Findings are outputted in the **diagnostics log** containing **intelligent insights**:

- **Root cause analysis:** Heavy locking is affecting the database performance. Affected queries are 0x9001RTYU and 0x9102EXZ4. Main blocking query is 0x9002FGJR on SPIDs 272. Consider stopping the blocking query.
- **Issue Status:** Active

The metrics used to measure and detect database performance issues are based on query duration, timeout requests, excessive wait times, and errored requests. For more information on metrics, see the [Detection metrics](#) section of this document.

Identified SQL Database performance degradations are recorded in the diagnostics log with intelligent entries that consist of the following properties:

PROPERTY	DETAILS
Database information	Metadata about a database on which an insight was detected, such as a resource URI.
Observed time range	Start and end time for the period of the detected insight.
Impacted metrics	Metrics that caused an insight to be generated: <ul style="list-style-type: none">• Query duration increase [seconds].• Excessive waiting [seconds].• Timed-out requests [percentage].• Errored-out requests [percentage].
Impact value	Value of a metric measured.

PROPERTY	DETAILS
Impacted queries and error codes	Query hash or error code. These can be used to easily correlate to affected queries. Metrics that consist of either query duration increase, waiting time, timeout counts, or error codes are provided.
Detections	Detection identified at the database during the time of an event. There are 15 detection patterns. For more information, see Troubleshoot database performance issues with Intelligent Insights .
Root cause analysis	Root cause analysis of the issue identified in a human-readable format. Some insights might contain a performance improvement recommendation where possible.

For a hands-on overview on using Intelligent Insights with Azure SQL Analytics and for typical usage scenarios, see the embedded video:

Intelligent Insights shines in discovering and troubleshooting SQL Database performance issues. In order to use Intelligent Insights to troubleshoot SQL Database and Managed Instance database performance issues, see [Troubleshoot Azure SQL Database performance issues with Intelligent Insights](#).

Intelligent Insights options

Intelligent Insights options available in Azure SQL Database are:

INTELLIGENT INSIGHTS OPTION	SINGLE DATABASE AND POOLED DATABASE SUPPORT	INSTANCE DATABASE SUPPORT
Configure Intelligent Insights - Configure Intelligent Insights analysis for your databases.	Yes	Yes
Stream insights to Azure SQL Analytics -- Stream insights to Azure SQL Analytics monitoring solution for Azure SQL Database.	Yes	Yes
Stream insights to Event Hub - Stream insights to Event Hubs for further custom integrations.	Yes	Yes
Stream insights to Azure Storage - Stream insights to Azure Storage for further analysis and long term archival.	Yes	Yes

Configure Intelligent Insights

Output of the Intelligent Insights is a smart performance diagnostics log. This log can be consumed in several ways - through streaming it to Azure SQL Analytics, Azure Event Hubs and Azure storage, or a third party product.

- Use the product with [Azure SQL Analytics](#) to view insights through the user interface of the Azure portal. This is the integrated Azure solution, and the most typical way to view insights.
- Use the product with Azure Event Hubs for development of custom monitoring and alerting scenarios
- Use the product with Azure storage for custom application development, such as for example custom reporting, long-term data archival and so forth.

Integration of Intelligent Insights with other products Azure SQL Analytics, Azure Event Hub, Azure storage, or third party products for consumption is performed through first enabling Intelligent Insights logging (the "SQLInsights" log) in the Diagnostic settings blade of a database, and then configuring Intelligent Insights log data to be streamed into one of these products.

For more information on how to enable Intelligent Insights logging and to configure log data to be streamed to a consuming product, see [Azure SQL Database metrics and diagnostics logging](#).

Set up with Azure SQL Analytics

Azure SQL Analytics solution provides graphical user interface, reporting and alerting capabilities on database performance, along with the Intelligent Insights diagnostics log data.

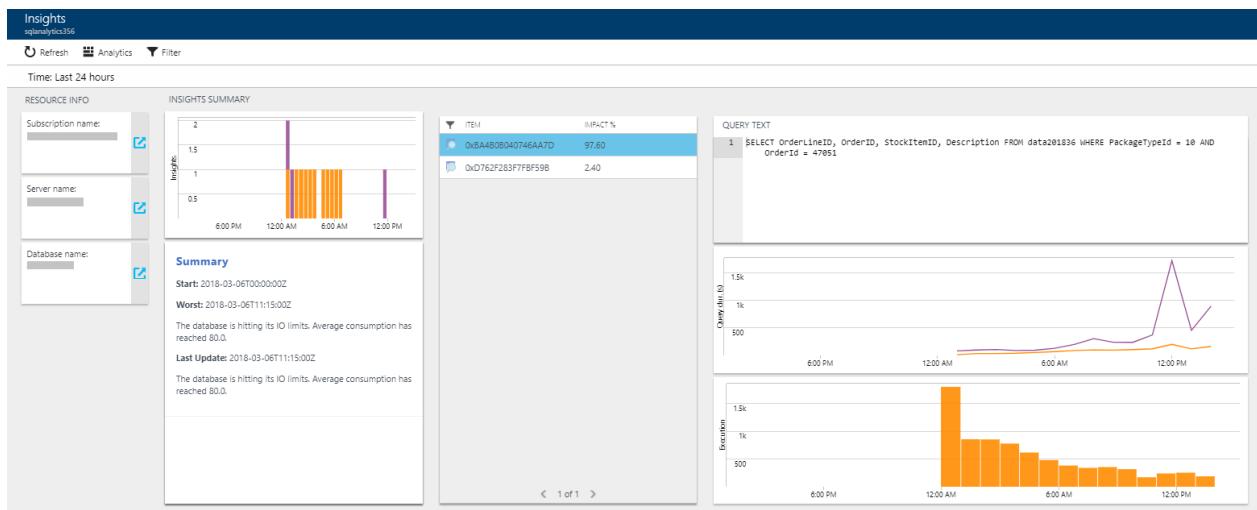
TIP

Quick getting started: The easiest way to get off the ground with using Intelligent Insights is to use it along with Azure SQL Analytics which will provide a graphical user interface to database performance issues. Add Azure SQL Analytics solution from the marketplace, create a workspace inside this solution, and then for each database you wish to enable Intelligent Insights on, configure streaming of "SQLInsights" log in the Diagnostics settings blade of a database to the workspace of Azure SQL Analytics.

Pre-requirement is to have Azure SQL Analytics added to your Azure portal dashboard from the marketplace and to create a workspace, see [configure Azure SQL Analytics](#)

To use Intelligent Insights with Azure SQL Analytics, configure Intelligent Insights log data to be streamed to Azure SQL Analytics workspace you've created in the previous step, see [Azure SQL Database metrics and diagnostics logging](#).

The following example shows an Intelligent Insights viewed through Azure SQL Analytics:



Set up with Event Hubs

To use Intelligent Insights with Event Hubs, configure Intelligent Insights log data to be streamed to Event Hubs, see [Stream Azure diagnostics logs to Event Hubs](#).

To use Event Hubs to setup custom monitoring and alerting, see [What to do with metrics and diagnostics logs in Event Hubs](#).

Set up with Azure Storage

To use Intelligent Insights with Storage, configure Intelligent Insights log data to be streamed to Storage, see [Stream into Azure Storage](#).

Custom integrations of Intelligent Insights log

To use Intelligent Insights with third party tools, or for custom alerting and monitoring development, see [Use the Intelligent Insights database performance diagnostics log](#).

Detection metrics

Metrics used for detection models that generate Intelligent Insights are based on monitoring:

- Query duration
- Timeout requests
- Excessive wait time
- Errorred out requests

Query duration and timeout requests are used as primary models in detecting issues with database workload performance. They're used because they directly measure what is happening with the workload. To detect all possible cases of workload performance degradation, excessive wait time and errored-out requests are used as additional models to indicate issues that affect the workload performance.

The system automatically considers changes to the workload and changes in the number of query requests made to the database to dynamically determine normal and out-of-the-ordinary database performance thresholds.

All of the metrics are considered together in various relationships through a scientifically derived data model that categorizes each performance issue detected. Information provided through an intelligent insight includes:

- Details of the performance issue detected.
- A root cause analysis of the issue detected.
- Recommendations on how to improve the performance of the monitored SQL database, where possible.

Query duration

The query duration degradation model analyzes individual queries and detects the increase in the time it takes to compile and execute a query compared to the performance baseline.

If SQL Database built-in intelligence detects a significant increase in query compile or query execution time that affects workload performance, these queries are flagged as query duration performance degradation issues.

The Intelligent Insights diagnostics log outputs the query hash of the query degraded in performance. The query hash indicates whether the performance degradation was related to query compile or execution time increase, which increased query duration time.

Timeout requests

The timeout requests degradation model analyzes individual queries and detects any increase in timeouts at the query execution level and the overall request timeouts at the database level compared to the performance baseline period.

Some of the queries might time out even before they reach the execution stage. Through the means of aborted workers vs. requests made, SQL Database built-in intelligence measures and analyzes all queries that reached the database whether they got to the execution stage or not.

After the number of timeouts for executed queries or the number of aborted request workers crosses the system-managed threshold, a diagnostics log is populated with intelligent insights.

The insights generated contain the number of timed-out requests and the number of timed-out queries. Indication of the performance degradation is related to timeout increase at the execution stage, or the overall database level is provided. When the increase in timeouts is deemed significant to database performance, these queries are flagged as timeout performance degradation issues.

Excessive wait times

The excessive wait time model monitors individual database queries. It detects unusually high query wait stats that crossed the system-managed absolute thresholds. The following query excessive wait-time metrics are observed by using the new SQL Server feature, Query Store Wait Stats (sys.query_store_wait_stats):

- Reaching resource limits
- Reaching elastic pool resource limits
- Excessive number of worker or session threads
- Excessive database locking
- Memory pressure
- Other wait stats

Reaching resource limits or elastic pool resource limits denote that consumption of available resources on a subscription or in the elastic pool crossed absolute thresholds. These stats indicate workload performance degradation. An excessive number of worker or session threads denotes a condition in which the number of worker threads or sessions initiated crossed absolute thresholds. These stats indicate workload performance degradation.

Excessive database locking denotes a condition in which the count of locks on a database has crossed absolute thresholds. This stat indicates a workload performance degradation. Memory pressure is a condition in which the number of threads requesting memory grants crossed an absolute threshold. This stat indicates a workload performance degradation.

Other wait stats detection indicates a condition in which miscellaneous metrics measured through the Query Store Wait Stats crossed an absolute threshold. These stats indicate workload performance degradation.

After excessive wait times are detected, depending on the data available, the Intelligent Insights diagnostics log outputs hashes of the affecting and affected queries degraded in performance, details of the metrics that cause queries to wait in execution, and measured wait time.

Errored requests

The errored requests degradation model monitors individual queries and detects an increase in the number of queries that errored out compared to the baseline period. This model also monitors critical exceptions that crossed absolute thresholds managed by SQL Database built-in intelligence. The system automatically considers the number of query requests made to the database and accounts for any workload changes in the monitored period.

When the measured increase in errored requests relative to the overall number of requests made is deemed significant to workload performance, affected queries are flagged as errored requests performance degradation issues.

The Intelligent Insights log outputs the count of errored requests. It indicates whether the performance degradation was related to an increase in errored requests or to crossing a monitored critical exception threshold and measured time of the performance degradation.

If any of the monitored critical exceptions cross the absolute thresholds managed by the system, an intelligent insight is generated with critical exception details.

Next steps

- Learn how to troubleshoot SQL Database performance issues with Intelligent Insights.
- Use the [Intelligent Insights SQL Database performance diagnostics log](#).
- Learn how to [monitor SQL Database by using SQL Analytics](#).
- Learn how to [collect and consume log data from your Azure resources](#).

Monitor and improve performance

11/17/2019 • 2 minutes to read • [Edit Online](#)

Azure SQL Database identifies potential problems in your database and recommends actions that can improve performance of your workload by providing intelligent tuning actions and recommendations.

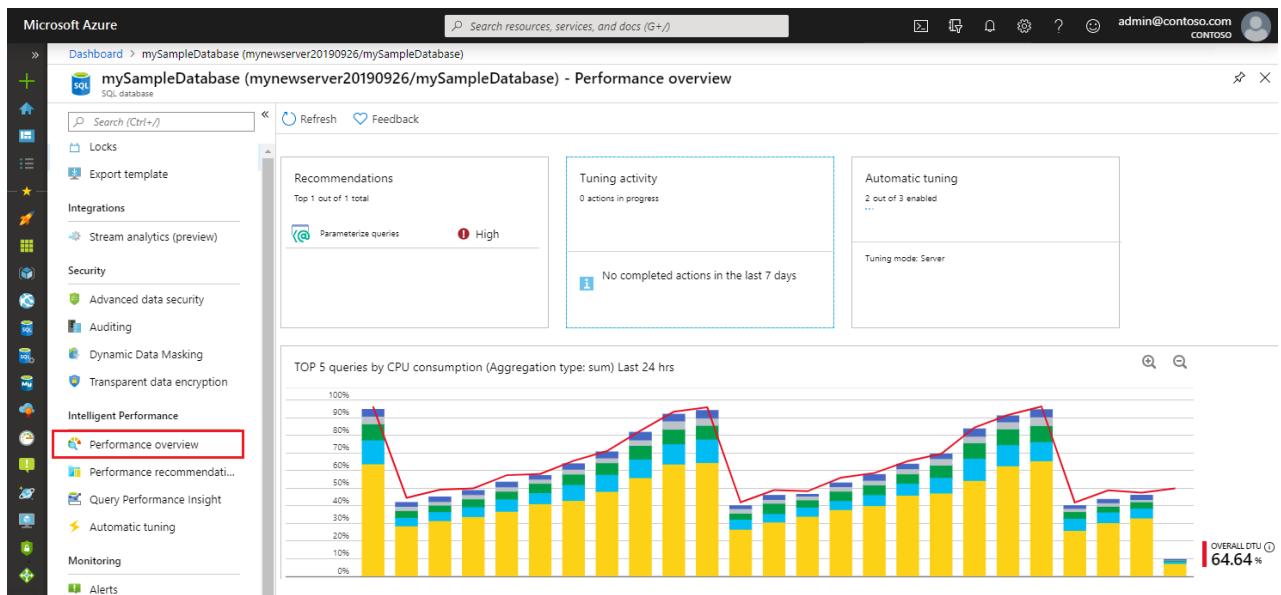
Performance tuning options

Performance tuning options available Azure SQL Database are available on the database navigation menu under "Intelligent Performance":

PERFORMANCE TUNING OPTION	SINGLE DATABASE AND POOLED DATABASE SUPPORT	INSTANCE DATABASE SUPPORT
Performance overview - Monitor all performance activities for your database.	Yes	No
Performance recommendations - Shows performance recommendations that can improve performance of your workload.	Yes	No
Query Performance Insight - Shows performance of top consuming queries on the database.	Yes	No
Automatic tuning - Use Azure SQL Database to automatically optimize your database performance.	Yes	No

Performance overview

This view provides a summary of your database performance, and helps you with performance tuning and troubleshooting.



- The **Recommendations** tile provides a breakdown of tuning recommendations for your database (top three recommendations are shown if there are more). Clicking this tile takes you to [Performance recommendations](#).
- The **Tuning activity** tile provides a summary of the ongoing and completed tuning actions for your database, giving you a quick view into the history of tuning activity. Clicking this tile takes you to the full tuning history view for your database.
- The **Auto-tuning** tile shows the [auto-tuning configuration](#) for your database (tuning options that are automatically applied to your database). Clicking this tile opens the automation configuration dialog.
- The **Database queries** tile shows the summary of the query performance for your database (overall DTU usage and top resource consuming queries). Clicking this tile takes you to [Query Performance Insight](#).

Performance recommendations

This page provides intelligent [tuning recommendations](#) that can improve your database's performance. The following types of recommendations are shown on this page:

- Recommendations on which indexes to create or drop.
- Recommendations when schema issues are identified in the database.
- Recommendations when queries can benefit from parameterized queries.

Action	Scope:	Reason:	Impact
Parameterize queries	Entire database	Non-parameterized queries are causing performance issues	High

Action	Recommendation description	Status	Time
Create index Initiated by: User	Table: [DataPoints] Indexed columns: [Name], [Money]	Success	5/9/2018 5:01:46 PM
Drop Index Initiated by: User	Index name: MyIndex321 Reason: Duplicate index	Success	4/25/2017 12:27:20 PM
Drop Index Initiated by: System	Index name: IX_FF Reason: Duplicate index	Success	4/24/2017 12:27:20 PM

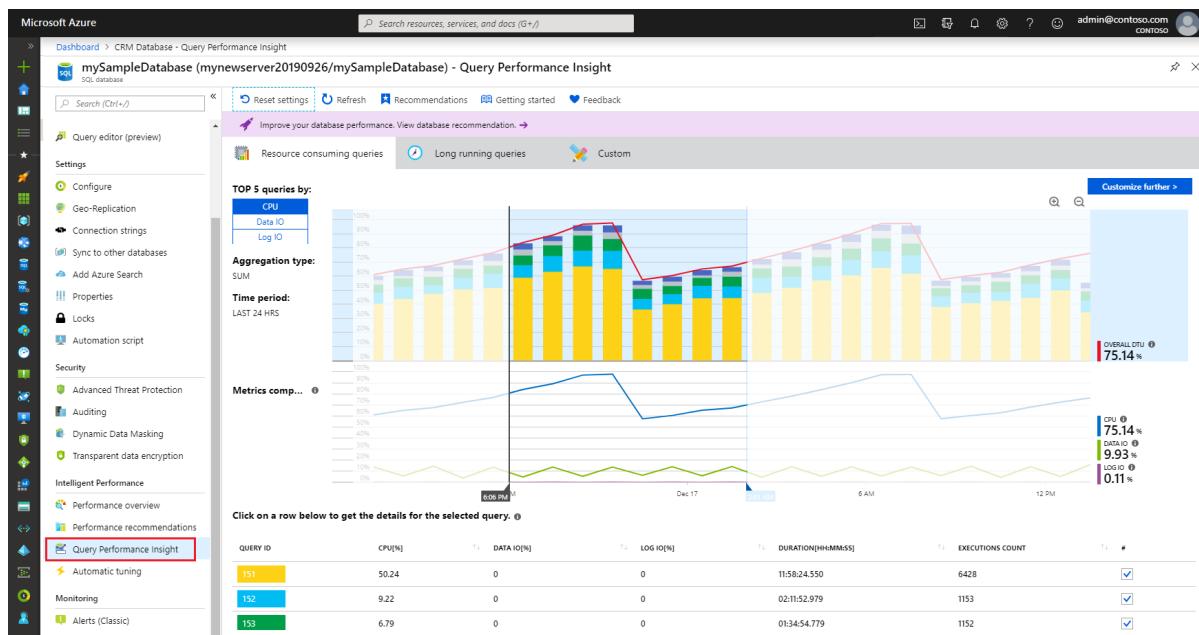
You can also find complete history of tuning actions that were applied in the past.

Learn how to find and apply performance recommendations in [Find and apply performance recommendations](#) article.

Query Performance Insight

[Query Performance Insight](#) allows you to spend less time troubleshooting database performance by providing:

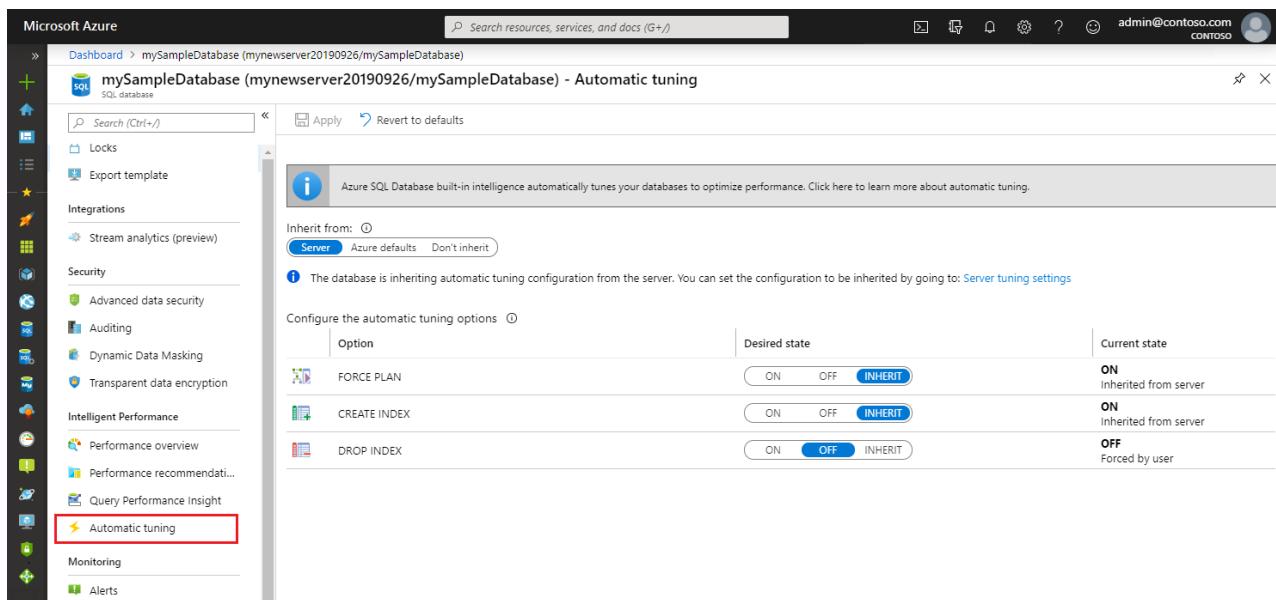
- Deeper insight into your databases resource (DTU) consumption.
- The top CPU consuming queries, which can potentially be tuned for improved performance.
- The ability to drill down into the details of a query.



Find more information about this page in the article [How to use Query Performance Insight](#).

Automatic tuning

Azure SQL databases can automatically tune database performance by applying [performance recommendations](#). To enable it, read [how to enable automatic tuning](#).



To learn more, read [Automatic tuning article](#).

Additional resources

- [Azure SQL Database performance guidance for single databases](#)
- [When should an elastic pool be used?](#)

Performance recommendations for SQL Database

11/13/2019 • 7 minutes to read • [Edit Online](#)

Azure SQL Database learns and adapts with your application. It provides customized recommendations that enable you to maximize the performance of your SQL databases. SQL Database continuously assesses and analyzes the usage history of your SQL databases. The recommendations that are provided are based on database-unique workload patterns and help improve performance.

TIP

[Automatic tuning](#) is the recommended method to automatically tune some of the most common database performance issues. [Query Performance Insights](#) is the recommended method for basic Azure SQL Database performance monitoring needs. [Azure SQL Analytics](#) is the recommended method for advanced monitoring of database performance at scale, with built-in intelligence for automated performance troubleshooting.

Performance recommendations options

Performance recommendation options available Azure SQL Database are:

PERFORMANCE RECOMMENDATION	SINGLE DATABASE AND POOLED DATABASE SUPPORT	INSTANCE DATABASE SUPPORT
Create index recommendations - Recommends creation of indexes that may improve performance of your workload.	Yes	No
Drop index recommendations - Recommends removal of redundant and duplicate indexes daily, except for unique indexes, and indexes that were not used for a long time (>90 days). Please note that this option is not compatible with applications using partition switching and index hints. Dropping unused indexes is not supported for Premium and Business Critical service tiers.	Yes	No
Parameterize queries recommendations (preview) - Recommends forced parametrization in cases when you have one or more queries that are constantly being recompiled but end up with the same query execution plan.	Yes	No

PERFORMANCE RECOMMENDATION	SINGLE DATABASE AND POOLED DATABASE SUPPORT	INSTANCE DATABASE SUPPORT
Fix schema issues recommendations (preview) - Recommendations for schema correction appear when the SQL Database service notices an anomaly in the number of schema-related SQL errors that are happening on your SQL database. Microsoft is currently deprecating "Fix schema issue" recommendations.	Yes	No

Create index recommendations

SQL Database continuously monitors the queries that are running and identifies the indexes that could improve performance. After there's enough confidence that a certain index is missing, a new **Create index** recommendation is created.

Azure SQL Database builds confidence by estimating the performance gain the index would bring through time. Depending on the estimated performance gain, recommendations are categorized as high, medium, or low.

Indexes that are created by using recommendations are always flagged as auto-created indexes. You can see which indexes are auto-created by looking at the sys.indexes view. Auto-created indexes don't block ALTER/RENAME commands.

If you try to drop the column that has an auto-created index over it, the command passes. The auto-created index is dropped with the command as well. Regular indexes block the ALTER/RENAME command on columns that are indexed.

After the create index recommendation is applied, Azure SQL Database compares the performance of the queries with the baseline performance. If the new index improved performance, the recommendation is flagged as successful and the impact report is available. If the index didn't improve performance, it's automatically reverted. SQL Database uses this process to ensure that recommendations improve database performance.

Any **create index** recommendation has a back-off policy that doesn't allow applying the recommendation if the resource usage of a database or pool is high. The back-off policy takes into account CPU, Data IO, Log IO, and available storage.

If CPU, Data IO, or Log IO is higher than 80% in the previous 30 minutes, the create index recommendation is postponed. If the available storage will be below 10% after the index is created, the recommendation goes into an error state. If, after a couple of days, automatic tuning still believes that the index would be beneficial, the process starts again.

This process repeats until there's enough available storage to create an index, or until the index isn't seen as beneficial anymore.

Drop index recommendations

Besides detecting missing indexes, SQL Database continuously analyzes the performance of existing indexes. If an index is not used, Azure SQL Database recommends dropping it. Dropping an index is recommended in two cases:

- The index is a duplicate of another index (same indexed and included column, partition schema, and filters).
- The index hasn't been used for a prolonged period (93 days).

Drop index recommendations also go through the verification after implementation. If the performance

improves, the impact report is available. If performance degrades, the recommendation is reverted.

Parameterize queries recommendations (preview)

Parameterize queries recommendations appear when you have one or more queries that are constantly being recompiled but end up with the same query execution plan. This condition creates an opportunity to apply forced parameterization. Forced parameterization, in turn, allows query plans to be cached and reused in the future, which improves performance and reduces resource usage.

Every query that's issued against SQL Server initially needs to be compiled to generate an execution plan. Each generated plan is added to the plan cache. Subsequent executions of the same query can reuse this plan from the cache, which eliminates the need for additional compilation.

Queries with non-parameterized values can lead to performance overhead because the execution plan is recompiled each time the non-parameterized values are different. In many cases, the same queries with different parameter values generate the same execution plans. These plans, however, are still separately added to the plan cache.

The process of recompiling execution plans uses database resources, increases the query duration time, and overflows the plan cache. These events, in turn, cause plans to be evicted from the cache. This SQL Server behavior can be altered by setting the forced parameterization option on the database.

To help you estimate the impact of this recommendation, you are provided with a comparison between the actual CPU usage and the projected CPU usage (as if the recommendation were applied). This recommendation can help you gain CPU savings. It can also help you decrease query duration and overhead for the plan cache, which means that more of the plans can stay in the cache and be reused. You can apply this recommendation quickly by selecting the **Apply** command.

After you apply this recommendation, it enables forced parameterization within minutes on your database. It starts the monitoring process, which lasts for approximately 24 hours. After this period, you can see the validation report. This report shows the CPU usage of your database 24 hours before and after the recommendation has been applied. SQL Database Advisor has a safety mechanism that automatically reverts the applied recommendation if performance regression has been detected.

Fix schema issues recommendations (preview)

IMPORTANT

Microsoft is currently deprecating "Fix schema issue" recommendations. We recommend that you use [Intelligent Insights](#) to monitor your database performance issues, including schema issues that the "Fix schema issue" recommendations previously covered.

Fix schema issues recommendations appear when the SQL Database service notices an anomaly in the number of schema-related SQL errors that are happening on your SQL database. This recommendation typically appears when your database encounters multiple schema-related errors (invalid column name, invalid object name, and so on) within an hour.

"Schema issues" are a class of syntax errors in SQL Server. They occur when the definition of the SQL query and the definition of the database schema aren't aligned. For example, one of the columns that's expected by the query might be missing in the target table or vice-versa.

The "Fix schema issue" recommendation appears when the Azure SQL Database service notices an anomaly in the number of schema-related SQL errors that are happening on your SQL database. The following table shows the errors that are related to schema issues:

SQL ERROR CODE	MESSAGE
201	Procedure or function " <i>expects parameter</i> ", which was not supplied.
207	Invalid column name '*'.
208	Invalid object name '*'.
213	Column name or number of supplied values does not match table definition.
2812	Could not find stored procedure '*'.
8144	Procedure or function * has too many arguments specified.

Custom applications

Developers might consider developing custom applications using performance recommendations for Azure SQL Database. All recommendations listed in the portal for a database can be accessed through [Get-AzSqlDatabaseRecommendedAction](#) API.

Next steps

Monitor your recommendations and continue to apply them to refine performance. Database workloads are dynamic and change continuously. SQL Database Advisor continues to monitor and provide recommendations that can potentially improve your database's performance.

- For more information about automatic tuning of database indexes and query execution plans, see [Azure SQL Database automatic tuning](#).
- For more information about automatically monitoring database performance with automated diagnostics and root cause analysis of performance issues, see [Azure SQL Intelligent Insights](#).
- For more information about how to use performance recommendations in the Azure portal, see [Performance recommendations in the Azure portal](#).
- See [Query Performance Insights](#) to learn about and view the performance impact of your top queries.

Manual tune query performance in Azure SQL Database

11/12/2019 • 16 minutes to read • [Edit Online](#)

Once you have identified a performance issue that you are facing with SQL Database, this article is designed to help you:

- Tune your application and apply some best practices that can improve performance.
- Tune the database by changing indexes and queries to more efficiently work with data.

This article assumes that you have already worked through the Azure SQL Database [database advisor recommendations](#) and the Azure SQL Database [auto-tuning recommendations](#). It also assumes that you have reviewed [An overview of monitoring and tuning](#) and its related articles related to troubleshooting performance issues. Additionally, this article assumes that you do not have a CPU resources, running-related performance issue that can be resolved by increasing the compute size or service tier to provide more resources to your database.

Tune your application

In traditional on-premises SQL Server, the process of initial capacity planning often is separated from the process of running an application in production. Hardware and product licenses are purchased first, and performance tuning is done afterward. When you use Azure SQL Database, it's a good idea to interweave the process of running an application and tuning it. With the model of paying for capacity on demand, you can tune your application to use the minimum resources needed now, instead of over-provisioning on hardware based on guesses of future growth plans for an application, which often are incorrect. Some customers might choose not to tune an application, and instead choose to over-provision hardware resources. This approach might be a good idea if you don't want to change a key application during a busy period. But, tuning an application can minimize resource requirements and lower monthly bills when you use the service tiers in Azure SQL Database.

Application characteristics

Although Azure SQL Database service tiers are designed to improve performance stability and predictability for an application, some best practices can help you tune your application to better take advantage of the resources at a compute size. Although many applications have significant performance gains simply by switching to a higher compute size or service tier, some applications need additional tuning to benefit from a higher level of service. For increased performance, consider additional application tuning for applications that have these characteristics:

- **Applications that have slow performance because of "chatty" behavior**

Chatty applications make excessive data access operations that are sensitive to network latency. You might need to modify these kinds of applications to reduce the number of data access operations to the SQL database. For example, you might improve application performance by using techniques like batching ad hoc queries or moving the queries to stored procedures. For more information, see [Batch queries](#).

- **Databases with an intensive workload that can't be supported by an entire single machine**

Databases that exceed the resources of the highest Premium compute size might benefit from scaling out the workload. For more information, see [Cross-database sharding](#) and [Functional partitioning](#).

- **Applications that have sub-optimal queries**

Applications, especially those in the data access layer, that have poorly tuned queries might not benefit from a higher compute size. This includes queries that lack a WHERE clause, have missing indexes, or have outdated statistics. These applications benefit from standard query performance-tuning techniques. For more information, see [Missing indexes](#) and [Query tuning and hinting](#).

- **Applications that have sub-optimal data access design**

Applications that have inherent data access concurrency issues, for example deadlocking, might not benefit from a higher compute size. Consider reducing round trips against the Azure SQL Database by caching data on the client side with the Azure Caching service or another caching technology. See [Application tier caching](#).

Tune your database

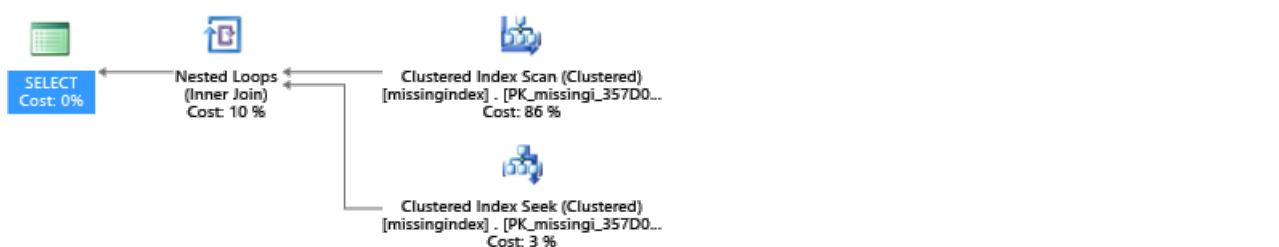
In this section, we look at some techniques that you can use to tune Azure SQL Database to gain the best performance for your application and run it at the lowest possible compute size. Some of these techniques match traditional SQL Server tuning best practices, but others are specific to Azure SQL Database. In some cases, you can examine the consumed resources for a database to find areas to further tune and extend traditional SQL Server techniques to work in Azure SQL Database.

Identifying and adding missing indexes

A common problem in OLTP database performance relates to the physical database design. Often, database schemas are designed and shipped without testing at scale (either in load or in data volume). Unfortunately, the performance of a query plan might be acceptable on a small scale but degrade substantially under production-level data volumes. The most common source of this issue is the lack of appropriate indexes to satisfy filters or other restrictions in a query. Often, missing indexes manifests as a table scan when an index seek could suffice.

In this example, the selected query plan uses a scan when a seek would suffice:

```
DROP TABLE dbo.missingindex;
CREATE TABLE dbo.missingindex (col1 INT IDENTITY PRIMARY KEY, col2 INT);
DECLARE @a int = 0;
SET NOCOUNT ON;
BEGIN TRANSACTION
    WHILE @a < 20000
        BEGIN
            INSERT INTO dbo.missingindex(col2) VALUES (@a);
            SET @a += 1;
        END
    COMMIT TRANSACTION;
    GO
SELECT m1.col1
    FROM dbo.missingindex m1 INNER JOIN dbo.missingindex m2 ON(m1.col1=m2.col1)
    WHERE m1.col2 = 4;
```



Azure SQL Database can help you find and fix common missing index conditions. DMVs that are built into Azure SQL Database look at query compilations in which an index would significantly reduce the estimated cost to run a query. During query execution, SQL Database tracks how often each query plan is executed, and tracks the estimated gap between the executing query plan and the imagined one where that index existed. You can use these DMVs to quickly guess which changes to your physical database design might improve overall workload

cost for a database and its real workload.

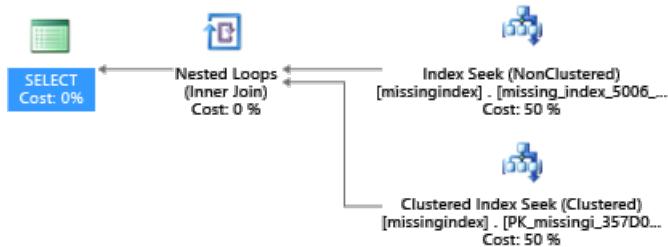
You can use this query to evaluate potential missing indexes:

```
SELECT
    CONVERT (varchar, getdate(), 126) AS runtime
    , mig.index_group_handle
    , mid.index_handle
    , CONVERT (decimal (28,1), migs.avg_total_user_cost * migs.avg_user_impact *
        (migs.user_seeks + migs.user_scans)) AS improvement_measure
    , 'CREATE INDEX missing_index_' + CONVERT (varchar, mig.index_group_handle) + '_' +
        CONVERT (varchar, mid.index_handle) + ' ON ' + mid.statement +
        (' + ISNULL (mid.equality_columns,'')
        + CASE WHEN mid.equality_columns IS NOT NULL
        AND mid.inequality_columns IS NOT NULL
        THEN ',' ELSE '' END + ISNULL (mid.inequality_columns, '') + ')'
        + ISNULL (' INCLUDE (' + mid.included_columns + ')', '') AS create_index_statement
    , migs.*
    , mid.database_id
    , mid.[object_id]
FROM sys.dm_db_missing_index_groups AS mig
    INNER JOIN sys.dm_db_missing_index_group_stats AS migs
        ON migs.group_handle = mig.index_group_handle
    INNER JOIN sys.dm_db_missing_index_details AS mid
        ON mig.index_handle = mid.index_handle
ORDER BY migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks + migs.user_scans) DESC
```

In this example, the query resulted in this suggestion:

```
CREATE INDEX missing_index_5006_5005 ON [dbo].[missingindex] ([col2])
```

After it's created, that same SELECT statement picks a different plan, which uses a seek instead of a scan, and then executes the plan more efficiently:



The key insight is that the IO capacity of a shared, commodity system is more limited than that of a dedicated server machine. There's a premium on minimizing unnecessary IO to take maximum advantage of the system in the DTU of each compute size of the Azure SQL Database service tiers. Appropriate physical database design choices can significantly improve the latency for individual queries, improve the throughput of concurrent requests handled per scale unit, and minimize the costs required to satisfy the query. For more information about the missing index DMVs, see [sys.dm_db_missing_index_details](#).

Query tuning and hinting

The query optimizer in Azure SQL Database is similar to the traditional SQL Server query optimizer. Most of the best practices for tuning queries and understanding the reasoning model limitations for the query optimizer also apply to Azure SQL Database. If you tune queries in Azure SQL Database, you might get the additional benefit of reducing aggregate resource demands. Your application might be able to run at a lower cost than an un-tuned equivalent because it can run at a lower compute size.

An example that is common in SQL Server and which also applies to Azure SQL Database is how the query optimizer "sniffs" parameters. During compilation, the query optimizer evaluates the current value of a parameter to determine whether it can generate a more optimal query plan. Although this strategy often can lead to a query

plan that is significantly faster than a plan compiled without known parameter values, currently it works imperfectly both in SQL Server and in Azure SQL Database. Sometimes the parameter is not sniffed, and sometimes the parameter is sniffed but the generated plan is sub-optimal for the full set of parameter values in a workload. Microsoft includes query hints (directives) so that you can specify intent more deliberately and override the default behavior of parameter sniffing. Often, if you use hints, you can fix cases in which the default SQL Server or Azure SQL Database behavior is imperfect for a specific customer workload.

The next example demonstrates how the query processor can generate a plan that is sub-optimal both for performance and resource requirements. This example also shows that if you use a query hint, you can reduce query run time and resource requirements for your SQL database:

```
DROP TABLE psptest1;
CREATE TABLE psptest1(col1 int primary key identity, col2 int, col3 binary(200));
DECLARE @a int = 0;
SET NOCOUNT ON;
BEGIN TRANSACTION
WHILE @a < 20000
BEGIN
    INSERT INTO psptest1(col2) values (1);
    INSERT INTO psptest1(col2) values (@a);
    SET @a += 1;
END
COMMIT TRANSACTION
CREATE INDEX i1 on psptest1(col2);
GO

CREATE PROCEDURE psp1 (@param1 int)
AS
BEGIN
    INSERT INTO t1 SELECT * FROM psptest1
    WHERE col2 = @param1
    ORDER BY col2;
END
GO

CREATE PROCEDURE psp2 (@param2 int)
AS
BEGIN
    INSERT INTO t1 SELECT * FROM psptest1 WHERE col2 = @param2
    ORDER BY col2
    OPTION (OPTIMIZE FOR (@param2 UNKNOWN))
END
GO

CREATE TABLE t1 (col1 int primary key, col2 int, col3 binary(200));
GO
```

The setup code creates a table that has skewed data distribution. The optimal query plan differs based on which parameter is selected. Unfortunately, the plan caching behavior doesn't always recompile the query based on the most common parameter value. So, it's possible for a sub-optimal plan to be cached and used for many values, even when a different plan might be a better plan choice on average. Then the query plan creates two stored procedures that are identical, except that one has a special query hint.

```
-- Prime Procedure Cache with scan plan
EXEC psp1 @param1=1;
TRUNCATE TABLE t1;

-- Iterate multiple times to show the performance difference
DECLARE @i int = 0;
WHILE @i < 1000
BEGIN
    EXEC psp1 @param1=2;
    TRUNCATE TABLE t1;
    SET @i += 1;
END
```

We recommend that you wait at least 10 minutes before you begin part 2 of the example, so that the results are distinct in the resulting telemetry data.

```
EXEC psp2 @param2=1;
TRUNCATE TABLE t1;

DECLARE @i int = 0;
WHILE @i < 1000
BEGIN
    EXEC psp2 @param2=2;
    TRUNCATE TABLE t1;
    SET @i += 1;
END
```

Each part of this example attempts to run a parameterized insert statement 1,000 times (to generate a sufficient load to use as a test data set). When it executes stored procedures, the query processor examines the parameter value that is passed to the procedure during its first compilation (parameter "sniffing"). The processor caches the resulting plan and uses it for later invocations, even if the parameter value is different. The optimal plan might not be used in all cases. Sometimes you need to guide the optimizer to pick a plan that is better for the average case rather than the specific case from when the query was first compiled. In this example, the initial plan generates a "scan" plan that reads all rows to find each value that matches the parameter:

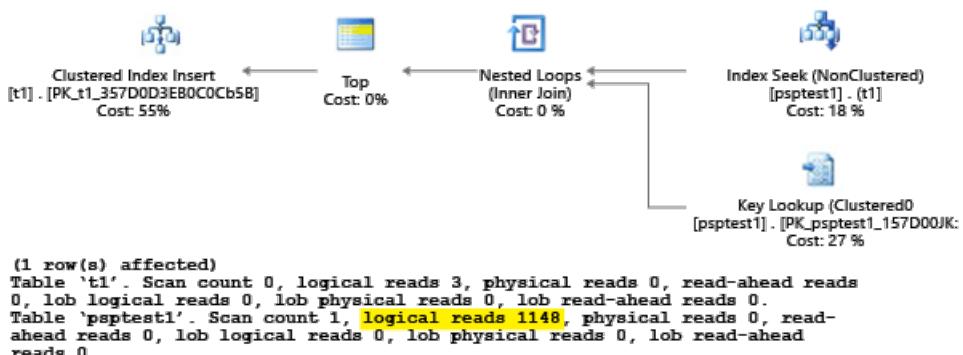


Because we executed the procedure by using the value 1, the resulting plan was optimal for the value 1 but was sub-optimal for all other values in the table. The result likely isn't what you would want if you were to pick each plan randomly, because the plan performs more slowly and uses more resources.

If you run the test with `SET STATISTICS IO` set to `ON`, the logical scan work in this example is done behind the scenes. You can see that there are 1,148 reads done by the plan (which is inefficient, if the average case is to return just one row):

```
(1 row(s) affected)
Table 't1'. Scan count 0, logical reads 3, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'psptest1'. Scan count 1, logical reads 1148, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

The second part of the example uses a query hint to tell the optimizer to use a specific value during the compilation process. In this case, it forces the query processor to ignore the value that is passed as the parameter, and instead to assume `UNKNOWN`. This refers to a value that has the average frequency in the table (ignoring skew). The resulting plan is a seek-based plan that is faster and uses fewer resources, on average, than the plan in part 1 of this example:



You can see the effect in the **sys.resource_stats** table (there is a delay from the time that you execute the test and when the data populates the table). For this example, part 1 executed during the 22:25:00 time window, and part 2 executed at 22:35:00. The earlier time window used more resources in that time window than the later one (because of plan efficiency improvements).

```

SELECT TOP 1000 *
FROM sys.resource_stats
WHERE database_name = 'resource1'
ORDER BY start_time DESC

```

	start_time	end_time	database_name	sku	storage_in_megabytes	avg_cpu_percent	avg_physical_data_read_percent	avg_log_write_percent
1	2013-06-18 22:45:00.000	2013-06-18 22:50:00.000	resource1	Basic	11.57	100.00	1.00	6.00
2	2013-06-18 22:40:00.000	2013-06-18 22:45:00.000	resource1	Basic	11.57	2.00	0.50	5.00
3	2013-06-18 22:35:00.000	2013-06-18 22:40:00.000	resource1	Basic	11.57	45.00	0.00	5.00
4	2013-06-18 22:30:00.000	2013-06-18 22:35:00.000	resource1	Basic	11.57	0.00	0.00	0.00
5	2013-06-18 22:25:00.000	2013-06-18 22:30:00.000	resource1	Basic	11.57	52.00	0.00	5.00
6	2013-06-18 22:20:00.000	2013-06-18 22:25:00.000	resource1	Basic	11.57	0.00	0.00	0.00

NOTE

Although the volume in this example is intentionally small, the effect of sub-optimal parameters can be substantial, especially on larger databases. The difference, in extreme cases, can be between seconds for fast cases and hours for slow cases.

You can examine **sys.resource_stats** to determine whether the resource for a test uses more or fewer resources than another test. When you compare data, separate the timing of tests so that they are not in the same 5-minute window in the **sys.resource_stats** view. The goal of the exercise is to minimize the total amount of resources used, and not to minimize the peak resources. Generally, optimizing a piece of code for latency also reduces resource consumption. Make sure that the changes you make to an application are necessary, and that the changes don't negatively affect the customer experience for someone who might be using query hints in the application.

If a workload has a set of repeating queries, often it makes sense to capture and validate the optimality of your plan choices because it drives the minimum resource size unit required to host the database. After you validate it, occasionally reexamine the plans to help you make sure that they have not degraded. You can learn more about [query hints \(Transact-SQL\)](#).

Cross-database sharding

Because Azure SQL Database runs on commodity hardware, the capacity limits for an individual database are lower than for a traditional on-premises SQL Server installation. Some customers use sharding techniques to spread database operations over multiple databases when the operations don't fit inside the limits of an individual database in Azure SQL Database. Most customers who use sharding techniques in Azure SQL Database split their data on a single dimension across multiple databases. For this approach, you need to understand that OLTP applications often perform transactions that apply to only one row or to a small group of rows in the schema.

NOTE

SQL Database now provides a library to assist with sharding. For more information, see [Elastic Database client library overview](#).

For example, if a database has customer name, order, and order details (like the traditional example Northwind database that ships with SQL Server), you could split this data into multiple databases by grouping a customer with the related order and order detail information. You can guarantee that the customer's data stays in an individual database. The application would split different customers across databases, effectively spreading the load across multiple databases. With sharding, customers not only can avoid the maximum database size limit, but Azure SQL Database also can process workloads that are significantly larger than the limits of the different compute sizes, as long as each individual database fits into its DTU.

Although database sharding doesn't reduce the aggregate resource capacity for a solution, it's highly effective at supporting very large solutions that are spread over multiple databases. Each database can run at a different compute size to support very large, "effective" databases with high resource requirements.

Functional partitioning

SQL Server users often combine many functions in an individual database. For example, if an application has logic to manage inventory for a store, that database might have logic associated with inventory, tracking purchase orders, stored procedures, and indexed or materialized views that manage end-of-month reporting. This technique makes it easier to administer the database for operations like backup, but it also requires you to size the hardware to handle the peak load across all functions of an application.

If you use a scale-out architecture in Azure SQL Database, it's a good idea to split different functions of an application into different databases. By using this technique, each application scales independently. As an application becomes busier (and the load on the database increases), the administrator can choose independent compute sizes for each function in the application. At the limit, with this architecture, an application can be larger than a single commodity machine can handle because the load is spread across multiple machines.

Batch queries

For applications that access data by using high-volume, frequent, ad hoc querying, a substantial amount of response time is spent on network communication between the application tier and the Azure SQL Database tier. Even when both the application and Azure SQL Database are in the same data center, the network latency between the two might be magnified by a large number of data access operations. To reduce the network round trips for the data access operations, consider using the option to either batch the ad hoc queries, or to compile them as stored procedures. If you batch the ad hoc queries, you can send multiple queries as one large batch in a single trip to Azure SQL Database. If you compile ad hoc queries in a stored procedure, you could achieve the same result as if you batch them. Using a stored procedure also gives you the benefit of increasing the chances of caching the query plans in Azure SQL Database so you can use the stored procedure again.

Some applications are write-intensive. Sometimes you can reduce the total IO load on a database by considering how to batch writes together. Often, this is as simple as using explicit transactions instead of auto-commit transactions in stored procedures and ad hoc batches. For an evaluation of different techniques you can use, see [Batching techniques for SQL Database applications in Azure](#). Experiment with your own workload to find the right model for batching. Be sure to understand that a model might have slightly different transactional consistency guarantees. Finding the right workload that minimizes resource use requires finding the right combination of consistency and performance trade-offs.

Application-tier caching

Some database applications have read-heavy workloads. Caching layers might reduce the load on the database and might potentially reduce the compute size required to support a database by using Azure SQL Database. With [Azure Cache for Redis](#), if you have a read-heavy workload, you can read the data once (or perhaps once per

application-tier machine, depending on how it is configured), and then store that data outside your SQL database. This is a way to reduce database load (CPU and read IO), but there is an effect on transactional consistency because the data being read from the cache might be out of sync with the data in the database. Although in many applications some level of inconsistency is acceptable, that's not true for all workloads. You should fully understand any application requirements before you implement an application-tier caching strategy.

Next steps

- For more information about DTU-based service tiers, see [DTU-based purchasing model](#).
- For more information about vCore-based service tiers, see [vCore-based purchasing model](#).
- For more information about elastic pools, see [What is an Azure elastic pool?](#)
- For information about performance and elastic pools, see [When to consider an elastic pool](#)

Monitoring performance Azure SQL Database using dynamic management views

11/7/2019 • 22 minutes to read • [Edit Online](#)

Microsoft Azure SQL Database enables a subset of dynamic management views to diagnose performance problems, which might be caused by blocked or long-running queries, resource bottlenecks, poor query plans, and so on. This topic provides information on how to detect common performance problems by using dynamic management views.

SQL Database partially supports three categories of dynamic management views:

- Database-related dynamic management views.
- Execution-related dynamic management views.
- Transaction-related dynamic management views.

For detailed information on dynamic management views, see [Dynamic Management Views and Functions \(Transact-SQL\)](#) in SQL Server Books Online.

Permissions

In SQL Database, querying a dynamic management view requires **VIEW DATABASE STATE** permissions. The **VIEW DATABASE STATE** permission returns information about all objects within the current database. To grant the **VIEW DATABASE STATE** permission to a specific database user, run the following query:

```
GRANT VIEW DATABASE STATE TO database_user;
```

In an instance of on-premises SQL Server, dynamic management views return server state information. In SQL Database, they return information regarding your current logical database only.

Identify CPU performance issues

If CPU consumption is above 80% for extended periods of time, consider the following troubleshooting steps:

The CPU issue is occurring now

If issue is occurring right now, there are two possible scenarios:

Many individual queries that cumulatively consume high CPU

Use the following query to identify top query hashes:

```

PRINT '-- top 10 Active CPU Consuming Queries (aggregated)--';
SELECT TOP 10 GETDATE() runtime,
       *
  FROM (SELECT query_stats.query_hash, SUM(query_stats.cpu_time) 'Total_Request_Cpu_Time_Ms', SUM(logical_reads)
        'Total_Request_Logical_Reads', MIN(start_time) 'Earliest_Request_start_Time', COUNT(*) 'Number_Of_Requests',
        SUBSTRING(REPLACE(REPLACE(MIN(query_stats.statement_text), CHAR(10), ' '), CHAR(13), ' '), 1, 256) AS
        "Statement_Text"
      FROM (SELECT req.*,
                  SUBSTRING(ST.text, (req.statement_start_offset / 2)+1, ((CASE statement_end_offset
WHEN -1 THEN DATALENGTH(ST.text)ELSE req.statement_end_offset END-req.statement_start_offset)/ 2)+1) AS
statement_text
            FROM sys.dm_exec_requests AS req
            CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) AS ST ) AS query_stats
      GROUP BY query_hash) AS t
  ORDER BY Total_Request_Cpu_Time_Ms DESC;

```

Long running queries that consume CPU are still running

Use the following query to identify these queries:

```

PRINT '--top 10 Active CPU Consuming Queries by sessions--';
SELECT TOP 10 req.session_id, req.start_time, cpu_time 'cpu_time_ms', OBJECT_NAME(ST.objectid, ST.dbid)
'ObjectName', SUBSTRING(REPLACE(REPLACE(SUBSTRING(ST.text, (req.statement_start_offset / 2)+1, ((CASE
statement_end_offset WHEN -1 THEN DATALENGTH(ST.text)ELSE req.statement_end_offset END-
req.statement_start_offset)/ 2)+1), CHAR(10), ' '), CHAR(13), ' '), 1, 512) AS statement_text
  FROM sys.dm_exec_requests AS req
  CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) AS ST
 ORDER BY cpu_time DESC;
GO

```

The CPU issue occurred in the past

If the issue occurred in the past and you want to do root cause analysis, use [Query Store](#). Users with database access can use T-SQL to query Query Store data. Query Store default configurations use a granularity of 1 hour. Use the following query to look at activity for high CPU consuming queries. This query returns the top 15 CPU consuming queries. Remember to change `rsi.start_time >= DATEADD(hour, -2, GETUTCDATE())`:

```

-- Top 15 CPU consuming queries by query hash
-- note that a query hash can have many query id if not parameterized or not parameterized properly
-- it grabs a sample query text by min
WITH AggregatedCPU AS (SELECT q.query_hash, SUM(count_executions * avg_cpu_time / 1000.0) AS
total_cpu_millisecond, SUM(count_executions * avg_cpu_time / 1000.0)/ SUM(count_executions) AS avg_cpu_millisecond,
MAX(rs.max_cpu_time / 1000.00) AS max_cpu_millisecond, MAX(max_logical_io_reads) max_logical_reads,
COUNT(DISTINCT p.plan_id) AS number_of_distinct_plans, COUNT(DISTINCT p.query_id) AS
number_of_distinct_query_ids, SUM(CASE WHEN rs.execution_type_desc='Aborted' THEN count_executions ELSE 0
END) AS Aborted_Execution_Count, SUM(CASE WHEN rs.execution_type_desc='Regular' THEN count_executions ELSE 0
END) AS Regular_Execution_Count, SUM(CASE WHEN rs.execution_type_desc='Exception' THEN count_executions ELSE
0 END) AS Exception_Execution_Count, SUM(count_executions) AS total_executions, MIN(qt.query_sql_text) AS
sampled_query_text
    FROM sys.query_store_query_text AS qt
        JOIN sys.query_store_query AS q ON qt.query_text_id=q.query_text_id
        JOIN sys.query_store_plan AS p ON q.query_id=p.query_id
        JOIN sys.query_store_runtime_stats AS rs ON rs.plan_id=p.plan_id
        JOIN sys.query_store_runtime_stats_interval AS rsi ON
rsi.runtime_stats_interval_id=rs.runtime_stats_interval_id
        WHERE rs.execution_type_desc IN ('Regular', 'Aborted', 'Exception')AND
rsi.start_time>=DATEADD(HOUR, -2, GETUTCDATE())
            GROUP BY q.query_hash), OrderedCPU AS (SELECT query_hash, total_cpu_millisecond,
avg_cpu_millisecond, max_cpu_millisecond, max_logical_reads, number_of_distinct_plans,
number_of_distinct_query_ids, total_executions, Aborted_Execution_Count, Regular_Execution_Count,
Exception_Execution_Count, sampled_query_text, ROW_NUMBER() OVER (ORDER BY total_cpu_millisecond DESC,
query_hash ASC) AS RN
                FROM AggregatedCPU)
SELECT OD.query_hash, OD.total_cpu_millisecond, OD.avg_cpu_millisecond, OD.max_cpu_millisecond, OD.max_logical_reads,
OD.number_of_distinct_plans, OD.number_of_distinct_query_ids, OD.total_executions,
OD.Aborted_Execution_Count, OD.Regular_Execution_Count, OD.Exception_Execution_Count, OD.sampled_query_text,
OD.RN
FROM OrderedCPU AS OD
WHERE OD.RN<=15
ORDER BY total_cpu_millisecond DESC;

```

Once you identify the problematic queries, it's time to tune those queries to reduce CPU utilization. If you don't have time to tune the queries, you may also choose to upgrade the SLO of the database to work around the issue.

Identify IO performance issues

When identifying IO performance issues, the top wait types associated with IO issues are:

- `PAGEIOLATCH_*`

For data file IO issues (including `PAGEIOLATCH_SH`, `PAGEIOLATCH_EX`, `PAGEIOLATCH_UP`). If the wait type name has **IO** in it, it points to an IO issue. If there is no **IO** in the page latch wait name, it points to a different type of problem (for example, tempdb contention).

- `WRITE_LOG`

For transaction log IO issues.

If the IO issue is occurring right now

Use the `sys.dm_exec_requests` or `sys.dm_os_waiting_tasks` to see the `wait_type` and `wait_time`.

Identify data and log IO usage

Use the following query to identify data and log IO usage. If the data or log IO is above 80%, it means users have used the available IO for the SQL DB service tier.

```

SELECT end_time, avg_data_io_percent, avg_log_write_percent
FROM sys.dm_db_resource_stats
ORDER BY end_time DESC;

```

If the IO limit has been reached, you have two options:

- Option 1: Upgrade the compute size or service tier
- Option 2: Identify and tune the queries consuming the most IO.

View buffer-related IO using the Query Store

For option 2, you can use the following query against Query Store for buffer-related IO to view the last two hours of tracked activity:

```
-- top queries that waited on buffer
-- note these are finished queries
WITH Aggregated AS (SELECT q.query_hash, SUM(total_query_wait_time_ms) total_wait_time_ms,
SUM(total_query_wait_time_ms / avg_query_wait_time_ms) AS total_executions, MIN(qt.query_sql_text) AS
sampled_query_text, MIN(wait_category_desc) AS wait_category_desc
    FROM sys.query_store_query_text AS qt
        JOIN sys.query_store_query AS q ON qt.query_text_id=q.query_text_id
        JOIN sys.query_store_plan AS p ON q.query_id=p.query_id
        JOIN sys.query_store_wait_stats AS waits ON waits.plan_id=p.plan_id
        JOIN sys.query_store_runtime_stats_interval AS rsi ON
rsi.runtime_stats_interval_id=waits.runtime_stats_interval_id
            WHERE wait_category_desc='Buffer IO' AND rsi.start_time>=DATEADD(HOUR, -2, GETUTCDATE())
            GROUP BY q.query_hash), Ordered AS (SELECT query_hash, total_executions,
total_wait_time_ms, sampled_query_text, wait_category_desc, ROW_NUMBER() OVER (ORDER BY total_wait_time_ms
DESC, query_hash ASC) AS RN
                FROM Aggregated)
SELECT OD.query_hash, OD.total_executions, OD.total_wait_time_ms, OD.sampled_query_text,
OD.wait_category_desc, OD.RN
FROM Ordered AS OD
WHERE OD.RN<=15
ORDER BY total_wait_time_ms DESC;
GO
```

View total log IO for WRITELOG waits

If the wait type is `WRITELOG`, use the following query to view total log IO by statement:

```
-- Top transaction log consumers
-- Adjust the time window by changing
-- rsi.start_time >= DATEADD(hour, -2, GETUTCDATE())
WITH AggregatedLogUsed
AS (SELECT q.query_hash,
    SUM(count_executions * avg_cpu_time / 1000.0) AS total_cpu_millisecond,
    SUM(count_executions * avg_cpu_time / 1000.0) / SUM(count_executions) AS avg_cpu_millisecond,
    SUM(count_executions * avg_log_bytes_used) AS total_log_bytes_used,
    MAX(rs.max_cpu_time / 1000.0) AS max_cpu_millisecond,
    MAX(max_logical_io_reads) max_logical_reads,
    COUNT(DISTINCT p.plan_id) AS number_of_distinct_plans,
    COUNT(DISTINCT p.query_id) AS number_of_distinct_query_ids,
    SUM( CASE
        WHEN rs.execution_type_desc = 'Aborted' THEN
            count_executions
        ELSE
            0
        END
    ) AS Aborted_Execution_Count,
    SUM( CASE
        WHEN rs.execution_type_desc = 'Regular' THEN
            count_executions
        ELSE
            0
        END
    ) AS Regular_Execution_Count,
    SUM( CASE
        WHEN rs.execution_type_desc = 'Exception' THEN
            count_executions
        ELSE
            0
        END
    ) AS Exception_Execution_Count
        FROM sys.query_store_runtime_stats AS rs
        JOIN sys.query_store_plan AS p ON rs.plan_id=p.plan_id
        JOIN sys.query_store_query AS q ON p.query_id=q.query_id
        JOIN sys.query_store_query_text AS qt ON q.query_text_id=qt.query_text_id
    WHERE rs.wait_type='WRITELOG'
    GROUP BY q.query_hash
    ORDER BY total_log_bytes_used DESC
    )
SELECT *
FROM AggregatedLogUsed
ORDER BY total_log_bytes_used DESC;
```

```

        ELSE
          0
        END
      ) AS Exception_Execution_Count,
      SUM(count_executions) AS total_executions,
      MIN(qt.query_sql_text) AS sampled_query_text
    FROM sys.query_store_query_text AS qt
    JOIN sys.query_store_query AS q
      ON qt.query_text_id = q.query_text_id
    JOIN sys.query_store_plan AS p
      ON q.query_id = p.query_id
    JOIN sys.query_store_runtime_stats AS rs
      ON rs.plan_id = p.plan_id
    JOIN sys.query_store_runtime_stats_interval AS rsi
      ON rsi.runtime_stats_interval_id = rs.runtime_stats_interval_id
    WHERE rs.execution_type_desc IN ( 'Regular', 'Aborted', 'Exception' )
      AND rsi.start_time >= DATEADD(HOUR, -2, GETUTCDATE())
    GROUP BY q.query_hash),
    OrderedLogUsed
  AS (SELECT query_hash,
    total_log_bytes_used,
    number_of_distinct_plans,
    number_of_distinct_query_ids,
    total_executions,
    Aborted_Execution_Count,
    Regular_Execution_Count,
    Exception_Execution_Count,
    sampled_query_text,
    ROW_NUMBER() OVER (ORDER BY total_log_bytes_used DESC, query_hash ASC) AS RN
  FROM AggregatedLogUsed)
  SELECT OD.total_log_bytes_used,
    OD.number_of_distinct_plans,
    OD.number_of_distinct_query_ids,
    OD.total_executions,
    OD.Aborted_Execution_Count,
    OD.Regular_Execution_Count,
    OD.Exception_Execution_Count,
    OD.sampled_query_text,
    OD.RN
  FROM OrderedLogUsed AS OD
  WHERE OD.RN <= 15
  ORDER BY total_log_bytes_used DESC;
  GO

```

Identify `tempdb` performance issues

When identifying IO performance issues, the top wait types associated with `tempdb` issues is `PAGELATCH_*` (not `PAGEIOLATCH_*`). However, `PAGELATCH_*` waits do not always mean you have `tempdb` contention. This wait may also mean that you have user-object data page contention due to concurrent requests targeting the same data page. To further confirm `tempdb` contention, use `sys.dm_exec_requests` to confirm that the `wait_resource` value begins with `2:x:y` where 2 is `tempdb` is the database ID, `x` is the file ID, and `y` is the page ID.

For tempdb contention, a common method is to reduce or re-write application code that relies on `tempdb`. Common `tempdb` usage areas include:

- Temp tables
- Table variables
- Table-valued parameters
- Version store usage (specifically associated with long running transactions)
- Queries that have query plans that use sorts, hash joins, and spools

Top queries that use table variables and temporary tables

Use the following query to identify top queries that use table variables and temporary tables:

```
SELECT plan_handle, execution_count, query_plan
INTO #tmpPlan
FROM sys.dm_exec_query_stats
    CROSS APPLY sys.dm_exec_query_plan(plan_handle);
GO

WITH XMLNAMESPACES('http://schemas.microsoft.com/sqlserver/2004/07/showplan' AS sp)
SELECT plan_handle, stmt.stmt_details.value('@Database', 'varchar(max)') 'Database',
stmt.stmt_details.value('@Schema', 'varchar(max)') 'Schema', stmt.stmt_details.value('@Table',
'varchar(max)') 'table'
INTO #tmp2
FROM(SELECT CAST(query_plan AS XML) sqlplan, plan_handle FROM #tmpPlan) AS p
    CROSS APPLY sqlplan.nodes('//sp:Object') AS stmt(stmt_details);
GO

SELECT t.plan_handle, [Database], [Schema], [table], execution_count
FROM(SELECT DISTINCT plan_handle, [Database], [Schema], [table]
    FROM #tmp2
    WHERE [table] LIKE '%@%' OR [table] LIKE '%##%') AS t
JOIN #tmpPlan AS t2 ON t.plan_handle=t2.plan_handle;
```

Identify long running transactions

Use the following query to identify long running transactions. Long running transactions prevent version store cleanup.

```

SELECT DB_NAME(dtr.database_id) 'database_name',
       sess.session_id,
       atr.name AS 'tran_name',
       atr.transaction_id,
       transaction_type,
       transaction_begin_time,
       database_transaction_begin_time transaction_state,
       is_user_transaction,
       sess.open_transaction_count,
       LTRIM(RTRIM(REPLACE(
                           REPLACE(
                               SUBSTRING(
                                   SUBSTRING(
                                       txt.text,
                                       (req.statement_start_offset / 2) + 1,
                                       ((CASE req.statement_end_offset
                                         WHEN -1 THEN
                                         DATALENGTH(txt.text)
                                         ELSE
                                         req.statement_end_offset
                                         END - req.statement_start_offset
                                         ) / 2
                                         ) + 1
                                         ),
                                         1,
                                         1000
                                         ),
                                         CHAR(10),
                                         ''
                                         ),
                                         CHAR(13),
                                         ''
                                         )
                                         )
                                         )
                                         ) Running_stmt_text,
recenttxt.text 'MostRecentSQLText'
FROM sys.dm_tran_active_transactions AS atr
INNER JOIN sys.dm_tran_database_transactions AS dtr
        ON dtr.transaction_id = atr.transaction_id
LEFT JOIN sys.dm_tran_session_transactions AS sess
        ON sess.transaction_id = atr.transaction_id
LEFT JOIN sys.dm_exec_requests AS req
        ON req.session_id = sess.session_id
        AND req.transaction_id = sess.transaction_id
LEFT JOIN sys.dm_exec_connections AS conn
        ON sess.session_id = conn.session_id
OUTER APPLY sys.dm_exec_sql_text(req.sql_handle) AS txt
OUTER APPLY sys.dm_exec_sql_text(conn.most_recent_sql_handle) AS recenttxt
WHERE atr.transaction_type != 2
      AND sess.session_id != @@spid
ORDER BY start_time ASC;

```

Identify memory grant wait performance issues

If your top wait type is `RESOURCE_SEMAHPORE` and you don't have a high CPU usage issue, you may have a memory grant waiting issue.

Determine if a `RESOURCE_SEMAHPORE` wait is a top wait

Use the following query to determine if a `RESOURCE_SEMAHPORE` wait is a top wait

```

SELECT wait_type,
       SUM(wait_time) AS total_wait_time_ms
  FROM sys.dm_exec_requests AS req
    JOIN sys.dm_exec_sessions AS sess
      ON req.session_id = sess.session_id
 WHERE is_user_process = 1
 GROUP BY wait_type
 ORDER BY SUM(wait_time) DESC;

```

Identify high memory-consuming statements

Use the following query to identify high memory-consuming statements:

```

SELECT IDENTITY(INT, 1, 1) rowId,
       CAST(query_plan AS XML) query_plan,
       p.query_id
  INTO #tmp
 FROM sys.query_store_plan AS p
   JOIN sys.query_store_runtime_stats AS r
     ON p.plan_id = r.plan_id
   JOIN sys.query_store_runtime_stats_interval AS i
     ON r.runtime_stats_interval_id = i.runtime_stats_interval_id
 WHERE start_time > '2018-10-11 14:00:00.0000000'
   AND end_time < '2018-10-17 20:00:00.0000000';
GO
;WITH cte
AS (SELECT query_id,
           query_plan,
           m.c.value('@SerialDesiredMemory', 'INT') AS SerialDesiredMemory
      FROM #tmp AS t
        CROSS APPLY t.query_plan.nodes('/**:MemoryGrantInfo[@SerialDesiredMemory[. > 0]]') AS m(c) )
SELECT TOP 50
       cte.query_id,
       t.query_sql_text,
       cte.query_plan,
       CAST(SerialDesiredMemory / 1024. AS DECIMAL(10, 2)) SerialDesiredMemory_MB
  FROM cte
   JOIN sys.query_store_query AS q
     ON cte.query_id = q.query_id
   JOIN sys.query_store_query_text AS t
     ON q.query_text_id = t.query_text_id
 ORDER BY SerialDesiredMemory DESC;

```

Identify the top 10 active memory grants

Use the following query to identify the top 10 active memory grants:

```

SELECT TOP 10
       CONVERT(VARCHAR(30), GETDATE(), 121) AS runtime,
       r.session_id,
       r.blocking_session_id,
       r.cpu_time,
       r.total_elapsed_time,
       r.reads,
       r.writes,
       r.logical_reads,
       r.row_count,
       wait_time,
       wait_type,
       r.command,
       OBJECT_NAME(txt.objectid, txt.dbid) 'Object_Name',
       LTRIM(RTRIM(REPLACE(
                           REPLACE(
                               SUBSTRING(
                                   SUBSTRING(

```

```

text,
(r.createStatement_start_offset / 2) + 1,
((CASE r.createStatement_end_offset
WHEN -1 THEN
    DATALENGTH(text)
ELSE
    r.createStatement_end_offset
END - r.createStatement_start_offset
) / 2
) + 1
),
1,
1000
),
CHAR(10),
'
),
CHAR(13),
'
)
)
) stmt_text,
mg.dop, --Degree of parallelism
mg.request_time, --Date and time when this query requested the
memory grant.
mg.grant_time, --NULL means memory has not been granted
mg.requested_memory_kb / 1024.0 requested_memory_mb, --Total requested amount of memory in megabytes
mg.granted_memory_kb / 1024.0 AS granted_memory_mb, --Total amount of memory actually granted in
megabytes. NULL if not granted
mg.required_memory_kb / 1024.0 AS required_memory_mb, --Minimum memory required to run this query in
megabytes.
max_used_memory_kb / 1024.0 AS max_used_memory_mb, --Estimated query cost.
mg.query_cost, --Time-out in seconds before this query gives up
mg.timeout_sec, the memory grant request.
mg.resource_semaphore_id, --Non-unique ID of the resource semaphore on
which this query is waiting.
mg.wait_time_ms, --Wait time in milliseconds. NULL if the memory
is already granted.
CASE mg.is_next_candidate --Is this process the next candidate for a memory grant
WHEN 1 THEN
    'Yes'
WHEN 0 THEN
    'No'
ELSE
    'Memory has been granted'
END AS 'Next Candidate for Memory Grant',
qp.query_plan
FROM sys.dm_exec_requests AS r
JOIN sys.dm_exec_query_memory_grants AS mg
ON r.session_id = mg.session_id
AND r.request_id = mg.request_id
CROSS APPLY sys.dm_exec_sql_text(mg.sql_handle) AS txt
CROSS APPLY sys.dm_exec_query_plan(r.plan_handle) AS qp
ORDER BY mg.granted_memory_kb DESC;

```

Calculating database and objects sizes

The following query returns the size of your database (in megabytes):

```
-- Calculates the size of the database.  
SELECT SUM(CAST(FILEPROPERTY(name, 'SpaceUsed') AS bigint) * 8192.) / 1024 / 1024 AS DatabaseSizeInMB  
FROM sys.database_files  
WHERE type_desc = 'ROWS';  
GO
```

The following query returns the size of individual objects (in megabytes) in your database:

```
-- Calculates the size of individual database objects.  
SELECT sys.objects.name, SUM(reserved_page_count) * 8.0 / 1024  
FROM sys.dm_db_partition_stats, sys.objects  
WHERE sys.dm_db_partition_stats.object_id = sys.objects.object_id  
GROUP BY sys.objects.name;  
GO
```

Monitoring connections

You can use the [sys.dm_exec_connections](#) view to retrieve information about the connections established to a specific Azure SQL Database server and the details of each connection. In addition, the [sys.dm_exec_sessions](#) view is helpful when retrieving information about all active user connections and internal tasks. The following query retrieves information on the current connection:

```
SELECT  
    c.session_id, c.net_transport, c.encrypt_option,  
    c.auth_scheme, s.host_name, s.program_name,  
    s.client_interface_name, s.login_name, s.nt_domain,  
    s.nt_user_name, s.original_login_name, c.connect_time,  
    s.login_time  
FROM sys.dm_exec_connections AS c  
JOIN sys.dm_exec_sessions AS s  
    ON c.session_id = s.session_id  
WHERE c.session_id = @@SPID;
```

NOTE

When executing the [sys.dm_exec_requests](#) and [sys.dm_exec_sessions](#) views, if you have **VIEW DATABASE STATE** permission on the database, you see all executing sessions on the database; otherwise, you see only the current session.

Monitor resource use

You can monitor resource usage using [SQL Database Query Performance Insight](#) and [Query Store](#).

You can also monitor usage using these two views:

- [sys.dm_db_resource_stats](#)
- [sys.resource_stats](#)

sys.dm_db_resource_stats

You can use the [sys.dm_db_resource_stats](#) view in every SQL database. The [sys.dm_db_resource_stats](#) view shows recent resource use data relative to the service tier. Average percentages for CPU, data IO, log writes, and memory are recorded every 15 seconds and are maintained for 1 hour.

Because this view provides a more granular look at resource use, use [sys.dm_db_resource_stats](#) first for any current-state analysis or troubleshooting. For example, this query shows the average and maximum resource use for the current database over the past hour:

```

SELECT
    AVG(avg_cpu_percent) AS 'Average CPU use in percent',
    MAX(avg_cpu_percent) AS 'Maximum CPU use in percent',
    AVG(avg_data_io_percent) AS 'Average data IO in percent',
    MAX(avg_data_io_percent) AS 'Maximum data IO in percent',
    AVG(avg_log_write_percent) AS 'Average log write use in percent',
    MAX(avg_log_write_percent) AS 'Maximum log write use in percent',
    AVG(avg_memory_usage_percent) AS 'Average memory use in percent',
    MAX(avg_memory_usage_percent) AS 'Maximum memory use in percent'
FROM sys.dm_db_resource_stats;

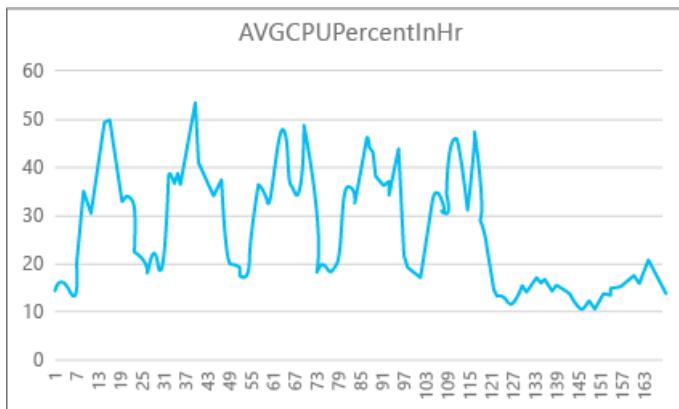
```

For other queries, see the examples in [sys.dm_db_resource_stats](#).

sys.resource_stats

The [sys.resource_stats](#) view in the **master** database has additional information that can help you monitor the performance of your SQL database at its specific service tier and compute size. The data is collected every 5 minutes and is maintained for approximately 14 days. This view is useful for a longer-term historical analysis of how your SQL database uses resources.

The following graph shows the CPU resource use for a Premium database with the P2 compute size for each hour in a week. This graph starts on a Monday, shows 5 work days, and then shows a weekend, when much less happens on the application.



From the data, this database currently has a peak CPU load of just over 50 percent CPU use relative to the P2 compute size (midday on Tuesday). If CPU is the dominant factor in the application's resource profile, then you might decide that P2 is the right compute size to guarantee that the workload always fits. If you expect an application to grow over time, it's a good idea to have an extra resource buffer so that the application doesn't ever reach the performance-level limit. If you increase the compute size, you can help avoid customer-visible errors that might occur when a database doesn't have enough power to process requests effectively, especially in latency-sensitive environments. An example is a database that supports an application that paints webpages based on the results of database calls.

Other application types might interpret the same graph differently. For example, if an application tries to process payroll data each day and has the same chart, this kind of "batch job" model might do fine at a P1 compute size. The P1 compute size has 100 DTUs compared to 200 DTUs at the P2 compute size. The P1 compute size provides half the performance of the P2 compute size. So, 50 percent of CPU use in P2 equals 100 percent CPU use in P1. If the application does not have timeouts, it might not matter if a job takes 2 hours or 2.5 hours to finish, if it gets done today. An application in this category probably can use a P1 compute size. You can take advantage of the fact that there are periods of time during the day when resource use is lower, so that any "big peak" might spill over into one of the troughs later in the day. The P1 compute size might be good for that kind of application (and save money), as long as the jobs can finish on time each day.

Azure SQL Database exposes consumed resource information for each active database in the [sys.resource_stats](#) view of the **master** database in each server. The data in the table is aggregated for 5-minute intervals. With the

Basic, Standard, and Premium service tiers, the data can take more than 5 minutes to appear in the table, so this data is more useful for historical analysis rather than near-real-time analysis. Query the **sys.resource_stats** view to see the recent history of a database and to validate whether the reservation you chose delivered the performance you want when needed.

NOTE

You must be connected to the **master** database of your SQL Database server to query **sys.resource_stats** in the following examples.

This example shows you how the data in this view is exposed:

```
SELECT TOP 10 *
FROM sys.resource_stats
WHERE database_name = 'resource1'
ORDER BY start_time DESC
```

	start_time	end_time	database_name	sku	storage_in_megabytes	avg_cpu_percent	avg_physical_data_read_percent	avg_log_write_percent
1	2013-05-18 22:45:00.00	2013-05-18 22:50:00.00	resource1	Basic	11.57	100.00	1.00	6.00
2	2013-05-18 22:40:00.00	2013-05-18 22:45:00.00	resource1	Basic	11.57	2.00	0.50	5.00
3	2013-05-18 22:35:00.00	2013-05-18 22:40:00.00	resource1	Basic	11.57	45.00	0.00	5.00
4	2013-05-18 22:30:00.00	2013-05-18 22:35:00.00	resource1	Basic	11.57	0.00	0.00	0.00
5	2013-05-18 22:25:00.00	2013-05-18 22:30:00.00	resource1	Basic	11.57	52.00	0.00	5.00
6	2013-05-18 22:20:00.00	2013-05-18 22:25:00.00	resource1	Basic	11.57	1.00	17.00	3.00
7	2013-05-18 22:15:00.00	2013-05-18 22:20:00.00	resource1	Basic	11.57	12.00	11.00	23.00
8	2013-05-18 22:10:00.00	2013-05-18 22:15:00.00	resource1	Basic	11.57	75.00	69.00	24.00
9	2013-05-18 22:05:00.00	2013-05-18 22:10:00.00	resource1	Basic	11.57	69.00	53.00	54.00
10	2013-05-18 22:00:00.00	2013-05-18 22:05:00.00	resource1	Basic	11.57	12.00	16.00	3.00

The next example shows you different ways that you can use the **sys.resource_stats** catalog view to get information about how your SQL database uses resources:

- To look at the past week's resource use for the database userdb1, you can run this query:

```
SELECT *
FROM sys.resource_stats
WHERE database_name = 'userdb1' AND
      start_time > DATEADD(day, -7, GETDATE())
ORDER BY start_time DESC;
```

- To evaluate how well your workload fits the compute size, you need to drill down into each aspect of the resource metrics: CPU, reads, writes, number of workers, and number of sessions. Here's a revised query using **sys.resource_stats** to report the average and maximum values of these resource metrics:

```
SELECT
    avg(avg_cpu_percent) AS 'Average CPU use in percent',
    max(avg_cpu_percent) AS 'Maximum CPU use in percent',
    avg(avg_data_io_percent) AS 'Average physical data IO use in percent',
    max(avg_data_io_percent) AS 'Maximum physical data IO use in percent',
    avg(avg_log_write_percent) AS 'Average log write use in percent',
    max(avg_log_write_percent) AS 'Maximum log write use in percent',
    avg(max_session_percent) AS 'Average % of sessions',
    max(max_session_percent) AS 'Maximum % of sessions',
    avg(max_worker_percent) AS 'Average % of workers',
    max(max_worker_percent) AS 'Maximum % of workers'
FROM sys.resource_stats
WHERE database_name = 'userdb1' AND start_time > DATEADD(day, -7, GETDATE());
```

- With this information about the average and maximum values of each resource metric, you can assess how well your workload fits into the compute size you chose. Usually, average values from **sys.resource_stats** give you a good baseline to use against the target size. It should be your primary measurement stick. For an example, you might be using the Standard service tier with S2 compute size. The average use

percentages for CPU and IO reads and writes are below 40 percent, the average number of workers is below 50, and the average number of sessions is below 200. Your workload might fit into the S1 compute size. It's easy to see whether your database fits in the worker and session limits. To see whether a database fits into a lower compute size with regards to CPU, reads, and writes, divide the DTU number of the lower compute size by the DTU number of your current compute size, and then multiply the result by 100:

$$S1 \text{ DTU} / S2 \text{ DTU} * 100 = 20 / 50 * 100 = 40$$

The result is the relative performance difference between the two compute sizes in percentage. If your resource use doesn't exceed this amount, your workload might fit into the lower compute size. However, you need to look at all ranges of resource use values, and determine, by percentage, how often your database workload would fit into the lower compute size. The following query outputs the fit percentage per resource dimension, based on the threshold of 40 percent that we calculated in this example:

```
SELECT
    (COUNT(database_name) - SUM(CASE WHEN avg_cpu_percent >= 40 THEN 1 ELSE 0 END) * 1.0) /
    COUNT(database_name) AS 'CPU Fit Percent',
    (COUNT(database_name) - SUM(CASE WHEN avg_log_write_percent >= 40 THEN 1 ELSE 0 END) * 1.0) /
    COUNT(database_name) AS 'Log Write Fit Percent',
    (COUNT(database_name) - SUM(CASE WHEN avg_data_io_percent >= 40 THEN 1 ELSE 0 END) * 1.0) /
    COUNT(database_name) AS 'Physical Data IO Fit Percent'
FROM sys.resource_stats
WHERE database_name = 'userdb1' AND start_time > DATEADD(day, -7, GETDATE());
```

Based on your database service tier, you can decide whether your workload fits into the lower compute size. If your database workload objective is 99.9 percent and the preceding query returns values greater than 99.9 percent for all three resource dimensions, your workload likely fits into the lower compute size.

Looking at the fit percentage also gives you insight into whether you should move to the next higher compute size to meet your objective. For example, userdb1 shows the following CPU use for the past week:

AVERAGE CPU PERCENT	MAXIMUM CPU PERCENT
24.5	100.00

The average CPU is about a quarter of the limit of the compute size, which would fit well into the compute size of the database. But, the maximum value shows that the database reaches the limit of the compute size. Do you need to move to the next higher compute size? Look at how many times your workload reaches 100 percent, and then compare it to your database workload objective.

```
SELECT
    (COUNT(database_name) - SUM(CASE WHEN avg_cpu_percent >= 100 THEN 1 ELSE 0 END) * 1.0) /
    COUNT(database_name) AS 'CPU fit percent'
    ,(COUNT(database_name) - SUM(CASE WHEN avg_log_write_percent >= 100 THEN 1 ELSE 0 END) * 1.0) /
    COUNT(database_name) AS 'Log write fit percent'
    ,(COUNT(database_name) - SUM(CASE WHEN avg_data_io_percent >= 100 THEN 1 ELSE 0 END) * 1.0) /
    COUNT(database_name) AS 'Physical data IO fit percent'
FROM sys.resource_stats
WHERE database_name = 'userdb1' AND start_time > DATEADD(day, -7, GETDATE());
```

If this query returns a value less than 99.9 percent for any of the three resource dimensions, consider either moving to the next higher compute size or use application-tuning techniques to reduce the load on the SQL database.

4. This exercise also considers your projected workload increase in the future.

For elastic pools, you can monitor individual databases in the pool with the techniques described in this section. But you can also monitor the pool as a whole. For information, see [Monitor and manage an elastic pool](#).

Maximum concurrent requests

To see the number of concurrent requests, run this Transact-SQL query on your SQL database:

```
```sql
SELECT COUNT(*) AS [Concurrent_Requests]
FROM sys.dm_exec_requests R;
````
```

To analyze the workload of an on-premises SQL Server database, modify this query to filter on the specific database you want to analyze. For example, if you have an on-premises database named MyDatabase, this Transact-SQL query returns the count of concurrent requests in that database:

```
```sql
SELECT COUNT(*) AS [Concurrent_Requests]
FROM sys.dm_exec_requests R
INNER JOIN sys.databases D ON D.database_id = R.database_id
AND D.name = 'MyDatabase';
````
```

This is just a snapshot at a single point in time. To get a better understanding of your workload and concurrent request requirements, you'll need to collect many samples over time.

Maximum concurrent logins

You can analyze your user and application patterns to get an idea of the frequency of logins. You also can run real-world loads in a test environment to make sure that you're not hitting this or other limits we discuss in this article. There isn't a single query or dynamic management view (DMV) that can show you concurrent login counts or history.

If multiple clients use the same connection string, the service authenticates each login. If 10 users simultaneously connect to a database by using the same username and password, there would be 10 concurrent logins. This limit applies only to the duration of the login and authentication. If the same 10 users connect to the database sequentially, the number of concurrent logins would never be greater than 1.

NOTE

Currently, this limit does not apply to databases in elastic pools.

Maximum sessions

To see the number of current active sessions, run this Transact-SQL query on your SQL database:

```
SELECT COUNT(*) AS [Sessions]
FROM sys.dm_exec_connections
```

If you're analyzing an on-premises SQL Server workload, modify the query to focus on a specific database. This query helps you determine possible session needs for the database if you are considering moving it to Azure SQL Database.

```

SELECT COUNT(*) AS [Sessions]
FROM sys.dm_exec_connections C
INNER JOIN sys.dm_exec_sessions S ON (S.session_id = C.session_id)
INNER JOIN sys.databases D ON (D.database_id = S.database_id)
WHERE D.name = 'MyDatabase'

```

Again, these queries return a point-in-time count. If you collect multiple samples over time, you'll have the best understanding of your session use.

For SQL Database analysis, you can get historical statistics on sessions by querying the [sys.resource_stats](#) view and reviewing the **active_session_count** column.

Monitoring query performance

Slow or long running queries can consume significant system resources. This section demonstrates how to use dynamic management views to detect a few common query performance problems. An older but still helpful reference for troubleshooting, is the [Troubleshooting Performance Problems in SQL Server 2008](#) article on Microsoft TechNet.

Finding top N queries

The following example returns information about the top five queries ranked by average CPU time. This example aggregates the queries according to their query hash, so that logically equivalent queries are grouped by their cumulative resource consumption.

```

```sql
SELECT TOP 5 query_stats.query_hash AS "Query Hash",
 SUM(query_stats.total_worker_time) / SUM(query_stats.execution_count) AS "Avg CPU Time",
 MIN(query_stats.statement_text) AS "Statement Text"
FROM
 (SELECT QS.*,
 SUBSTRING(ST.text, (QS.statement_start_offset/2) + 1,
 ((CASE statement_end_offset
 WHEN -1 THEN DATALENGTH(ST.text)
 ELSE QS.statement_end_offset END
 - QS.statement_start_offset)/2) + 1) AS statement_text
 FROM sys.dm_exec_query_stats AS QS
 CROSS APPLY sys.dm_exec_sql_text(QS.sql_handle) as ST) as query_stats
GROUP BY query_stats.query_hash
ORDER BY 2 DESC;
```

```

Monitoring blocked queries

Slow or long-running queries can contribute to excessive resource consumption and be the consequence of blocked queries. The cause of the blocking can be poor application design, bad query plans, the lack of useful indexes, and so on. You can use the [sys.dm_tran_locks](#) view to get information about the current locking activity in your Azure SQL Database. For example code, see [sys.dm_tran_locks \(Transact-SQL\)](#) in SQL Server Books Online.

Monitoring query plans

An inefficient query plan also may increase CPU consumption. The following example uses the [sys.dm_exec_query_stats](#) view to determine which query uses the most cumulative CPU.

```
```sql
SELECT
 highest_cpu_queries.plan_handle,
 highest_cpu_queries.total_worker_time,
 q.dbid,
 q.objectid,
 q.number,
 q.encrypted,
 q.[text]
FROM
 (SELECT TOP 50
 qs.plan_handle,
 qs.total_worker_time
 FROM
 sys.dm_exec_query_stats qs
 ORDER BY qs.total_worker_time desc) AS highest_cpu_queries
CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS q
ORDER BY highest_cpu_queries.total_worker_time DESC;
```

```

See also

[Introduction to SQL Database](#)

Operating the Query Store in Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

Query Store in Azure is a fully managed database feature that continuously collects and presents detailed historic information about all queries. You can think about Query Store as similar to an airplane's flight data recorder that significantly simplifies query performance troubleshooting both for cloud and on-premises customers. This article explains specific aspects of operating Query Store in Azure. Using this pre-collected query data, you can quickly diagnose and resolve performance problems and thus spend more time focusing on their business.

Query Store has been [globally available](#) in Azure SQL Database since November, 2015. Query Store is the foundation for performance analysis and tuning features, such as [SQL Database Advisor and Performance Dashboard](#). At the moment of publishing this article, Query Store is running in more than 200,000 user databases in Azure, collecting query-related information for several months, without interruption.

IMPORTANT

Microsoft is in the process of activating Query Store for all Azure SQL databases (existing and new).

Optimal Query Store Configuration

This section describes optimal configuration defaults that are designed to ensure reliable operation of the Query Store and dependent features, such as [SQL Database Advisor and Performance Dashboard](#). Default configuration is optimized for continuous data collection, that is minimal time spent in OFF/READ_ONLY states.

| CONFIGURATION | DESCRIPTION | DEFAULT | COMMENT |
|----------------------------|--|---------|--|
| MAX_STORAGE_SIZE_MB | Specifies the limit for the data space that Query Store can take inside the customer database | 100 | Enforced for new databases |
| INTERVAL_LENGTH_MINUTE_S | Defines size of time window during which collected runtime statistics for query plans are aggregated and persisted. Every active query plan has at most one row for a period of time defined with this configuration | 60 | Enforced for new databases |
| STALE_QUERY_THRESHOLD_DAYS | Time-based cleanup policy that controls the retention period of persisted runtime statistics and inactive queries | 30 | Enforced for new databases and databases with previous default (367) |
| SIZE_BASED_CLEANUP_MODE | Specifies whether automatic data cleanup takes place when Query Store data size approaches the limit | AUTO | Enforced for all databases |

| Configuration | Description | Default | Comment |
|------------------------|---|---------|----------------------------|
| QUERY_CAPTURE_MODE | Specifies whether all queries or only a subset of queries are tracked | AUTO | Enforced for all databases |
| FLUSH_INTERVAL_SECONDS | Specifies maximum period during which captured runtime statistics are kept in memory, before flushing to disk | 900 | Enforced for new databases |
| | | | |

IMPORTANT

These defaults are automatically applied in the final stage of Query Store activation in all Azure SQL databases (see preceding important note). After this light up, Azure SQL Database won't be changing configuration values set by customers, unless they negatively impact primary workload or reliable operations of the Query Store.

If you want to stay with your custom settings, use [ALTER DATABASE with Query Store options](#) to revert configuration to the previous state. Check out [Best Practices with the Query Store](#) in order to learn how to choose optimal configuration parameters.

Next steps

[SQL Database Performance Insight](#)

Additional resources

For more information check out the following articles:

- [A flight data recorder for your database](#)
- [Monitoring Performance By Using the Query Store](#)
- [Query Store Usage Scenarios](#)

Troubleshoot Azure SQL Database performance issues with Intelligent Insights

2/24/2020 • 24 minutes to read • [Edit Online](#)

This page provides information on Azure SQL Database and Managed Instance performance issues detected through the [Intelligent Insights](#) database performance diagnostics log. The diagnostic log telemetry can be streamed to [Azure Monitor logs](#), [Azure Event Hubs](#), [Azure Storage](#), or a third-party solution for custom DevOps alerting and reporting capabilities.

NOTE

For a quick SQL Database performance troubleshooting guide using Intelligent Insights, see the [Recommended troubleshooting flow](#) flowchart in this document.

Detectable database performance patterns

Intelligent Insights automatically detects performance issues with SQL Database and Managed Instance databases based on query execution wait times, errors, or time-outs. It outputs detected performance patterns to the diagnostics log. Detectable performance patterns are summarized in the table below.

| DETECTABLE PERFORMANCE PATTERNS | DESCRIPTION FOR AZURE SQL DATABASE AND ELASTIC POOLS | DESCRIPTION FOR DATABASES IN MANAGED INSTANCE |
|---------------------------------|--|--|
| Reaching resource limits | Consumption of available resources (DTUs), database worker threads, or database login sessions available on the monitored subscription has reached limits. This is affecting the SQL Database performance. | Consumption of CPU resources is reaching Managed Instance limits. This is affecting the database performance. |
| Workload increase | Workload increase or continuous accumulation of workload on the database was detected. This is affecting the SQL Database performance. | Workload increase has been detected. This is affecting the database performance. |
| Memory pressure | Workers that requested memory grants have to wait for memory allocations for statistically significant amounts of time, or an increased accumulation of workers that requested memory grants exists. This is affecting the SQL Database performance. | Workers that have requested memory grants are waiting for memory allocations for a statistically significant amount of time. This is affecting the database performance. |
| Locking | Excessive database locking was detected affecting the SQL Database performance. | Excessive database locking was detected affecting the database performance. |
| Increased MAXDOP | The maximum degree of parallelism option (MAXDOP) has changed affecting the query execution efficiency. This is affecting the SQL Database performance. | The maximum degree of parallelism option (MAXDOP) has changed affecting the query execution efficiency. This is affecting the database performance. |

| DETECTABLE PERFORMANCE PATTERNS | DESCRIPTION FOR AZURE SQL DATABASE AND ELASTIC POOLS | DESCRIPTION FOR DATABASES IN MANAGED INSTANCE |
|--|---|---|
| Pagelatch contention | Multiple threads are concurrently attempting to access the same in-memory data buffer pages resulting in increased wait times and causing pagelatch contention. This is affecting the SQL database performance. | Multiple threads are concurrently attempting to access the same in-memory data buffer pages resulting in increased wait times and causing pagelatch contention. This is affecting database the performance. |
| Missing Index | Missing index was detected affecting the SQL database performance. | Missing index was detected affecting the database performance. |
| New Query | New query was detected affecting the overall SQL Database performance. | New query was detected affecting the overall database performance. |
| Increased Wait Statistic | Increased database wait times were detected affecting the SQL database performance. | Increased database wait times were detected affecting the database performance. |
| TempDB Contention | Multiple threads are trying to access the same TempDB resource causing a bottleneck. This is affecting the SQL Database performance. | Multiple threads are trying to access the same TempDB resource causing a bottleneck. This is affecting the database performance. |
| Elastic pool DTU shortage | Shortage of available eDTUs in the elastic pool is affecting SQL Database performance. | Not available for Managed Instance as it uses vCore model. |
| Plan Regression | New plan, or a change in the workload of an existing plan was detected. This is affecting the SQL Database performance. | New plan, or a change in the workload of an existing plan was detected. This is affecting the database performance. |
| Database-scoped configuration value change | Configuration change on the SQL Database was detected affecting the database performance. | Configuration change on the database was detected affecting the database performance. |
| Slow client | Slow application client is unable to consume output from the database fast enough. This is affecting the SQL Database performance. | Slow application client is unable to consume output from the database fast enough. This is affecting the database performance. |
| Pricing tier downgrade | Pricing tier downgrade action decreased available resources. This is affecting the SQL Database performance. | Pricing tier downgrade action decreased available resources. This is affecting the database performance. |

TIP

For continuous performance optimization of SQL Database, enable [Azure SQL Database automatic tuning](#). This unique feature of SQL Database built-in intelligence continuously monitors your SQL database, automatically tunes indexes, and applies query execution plan corrections.

The following section describes detectable performance patterns in more detail.

Reaching resource limits

What is happening

This detectable performance pattern combines performance issues that are related to reaching available resource limits, worker limits, and session limits. After this performance issue is detected, a description field of the diagnostics log indicates whether the performance issue is related to resource, worker, or session limits.

Resources on SQL Database are typically referred to [DTU](#) or [vCore](#) resources. The pattern of reaching resource limits is recognized when detected query performance degradation is caused by reaching any of the measured resource limits.

The session limits resource denotes the number of available concurrent logins to the SQL database. This performance pattern is recognized when applications that are connected to the SQL databases have reached the number of available concurrent logins to the database. If applications attempt to use more sessions than are available on a database, the query performance is affected.

Reaching worker limits is a specific case of reaching resource limits because available workers aren't counted in the DTU or vCore usage. Reaching worker limits on a database can cause the rise of resource-specific wait times, which results in query performance degradation.

Troubleshooting

The diagnostics log outputs query hashes of queries that affected the performance and resource consumption percentages. You can use this information as a starting point for optimizing your database workload. In particular, you can optimize the queries that affect the performance degradation by adding indexes. Or you can optimize applications with a more even workload distribution. If you're unable to reduce workloads or make optimizations, consider increasing the pricing tier of your SQL database subscription to increase the amount of resources available.

If you have reached the available session limits, you can optimize your applications by reducing the number of logins made to the database. If you're unable to reduce the number of logins from your applications to the database, consider increasing the pricing tier of your database. Or you can split and move your database into multiple databases for a more balanced workload distribution.

For more suggestions on resolving session limits, see [How to deal with the limits of SQL Database maximum logins](#). See [Overview of resource limits on a SQL Database server](#) for information about limits at the server and subscription levels.

Workload Increase

What is happening

This performance pattern identifies issues caused by a workload increase or, in its more severe form, a workload pile-up.

This detection is made through a combination of several metrics. The basic metric measured is detecting an increase in workload compared with the past workload baseline. The other form of detection is based on measuring a large increase in active worker threads that is large enough to affect the query performance.

In its more severe form, the workload might continuously pile up due to the inability of the SQL database to handle the workload. The result is a continuously growing workload size, which is the workload pile-up condition. Due to this condition, the time that the workload waits for execution grows. This condition represents one of the most severe database performance issues. This issue is detected through monitoring the increase in the number of aborted worker threads.

Troubleshooting

The diagnostics log outputs the number of queries whose execution has increased and the query hash of the query with the largest contribution to the workload increase. You can use this information as a starting point for optimizing the workload. The query identified as the largest contributor to the workload increase is especially

useful as your starting point.

You might consider distributing the workloads more evenly to the database. Consider optimizing the query that is affecting the performance by adding indexes. You also might distribute your workload among multiple databases. If these solutions aren't possible, consider increasing the pricing tier of your SQL database subscription to increase the amount of resources available.

Memory Pressure

What is happening

This performance pattern indicates degradation in the current database performance caused by memory pressure, or in its more severe form a memory pile-up condition, compared to the past seven-day performance baseline.

Memory pressure denotes a performance condition in which there is a large number of worker threads requesting memory grants in the SQL database. The high volume causes a high memory utilization condition in which the SQL database is unable to efficiently allocate memory to all workers that request it. One of the most common reasons for this issue is related to the amount of memory available to the SQL database on one hand. On the other hand, an increase in workload causes the increase in worker threads and the memory pressure.

The more severe form of memory pressure is the memory pile-up condition. This condition indicates that a higher number of worker threads are requesting memory grants than there are queries releasing the memory. This number of worker threads requesting memory grants also might be continuously increasing (piling up) because the SQL database engine is unable to allocate memory efficiently enough to meet the demand. The memory pile-up condition represents one of the most severe database performance issues.

Troubleshooting

The diagnostics log outputs the memory object store details with the clerk (that is, worker thread) marked as the highest reason for high memory usage and relevant time stamps. You can use this information as the basis for troubleshooting.

You can optimize or remove queries related to the clerks with the highest memory usage. You also can make sure that you aren't querying data that you don't plan to use. Good practice is to always use a WHERE clause in your queries. In addition, we recommend that you create nonclustered indexes to seek the data rather than scan it.

You also can reduce the workload by optimizing or distributing it over multiple databases. Or you can distribute your workload among multiple databases. If these solutions aren't possible, consider increasing the pricing tier of your SQL database subscription to increase the amount of memory resources available to the database.

For additional troubleshooting suggestions, see [Memory grants meditation: The mysterious SQL Server memory consumer with many names](#).

Locking

What is happening

This performance pattern indicates degradation in the current database performance in which excessive database locking is detected compared to the past seven-day performance baseline.

In modern RDBMS, locking is essential for implementing multithreaded systems in which performance is maximized by running multiple simultaneous workers and parallel database transactions where possible. Locking in this context refers to the built-in access mechanism in which only a single transaction can exclusively access the rows, pages, tables, and files that are required and not compete with another transaction for resources. When the transaction that locked the resources for use is done with them, the lock on those resources is released, which allows other transactions to access required resources. For more information on locking, see [Lock in the database engine](#).

If transactions executed by the SQL engine are waiting for prolonged periods of time to access resources locked for use, this wait time causes the slowdown of the workload execution performance.

Troubleshooting

The diagnostics log outputs locking details that you can use as the basis for troubleshooting. You can analyze the reported blocking queries, that is, the queries that introduce the locking performance degradation, and remove them. In some cases, you might be successful in optimizing the blocking queries.

The simplest and safest way to mitigate the issue is to keep transactions short and to reduce the lock footprint of the most expensive queries. You can break up a large batch of operations into smaller operations. Good practice is to reduce the query lock footprint by making the query as efficient as possible. Reduce large scans because they increase chances of deadlocks and adversely affect overall database performance. For identified queries that cause locking, you can create new indexes or add columns to the existing index to avoid the table scans.

For more suggestions, see [How to resolve blocking problems that are caused by lock escalation in SQL Server](#).

Increased MAXDOP

What is happening

This detectable performance pattern indicates a condition in which a chosen query execution plan was parallelized more than it should have been. The SQL Database query optimizer can enhance the workload performance by executing queries in parallel to speed up things where possible. In some cases, parallel workers processing a query spend more time waiting on each other to synchronize and merge results compared to executing the same query with fewer parallel workers, or even in some cases compared to a single worker thread.

The expert system analyzes the current database performance compared to the baseline period. It determines if a previously running query is running slower than before because the query execution plan is more parallelized than it should be.

The MAXDOP server configuration option on SQL Database is used to control how many CPU cores can be used to execute the same query in parallel.

Troubleshooting

The diagnostics log outputs query hashes related to queries for which the duration of execution increased because they were parallelized more than they should have been. The log also outputs CXP wait times. This time represents the time a single organizer/coordinator thread (thread 0) is waiting for all other threads to finish before merging the results and moving ahead. In addition, the diagnostics log outputs the wait times that the poor-performing queries were waiting in execution overall. You can use this information as the basis for troubleshooting.

First, optimize or simplify complex queries. Good practice is to break up long batch jobs into smaller ones. In addition, ensure that you created indexes to support your queries. You can also manually enforce the maximum degree of parallelism (MAXDOP) for a query that was flagged as poor performing. To configure this operation by using T-SQL, see [Configure the MAXDOP server configuration option](#).

Setting the MAXDOP server configuration option to zero (0) as a default value denotes that SQL Database can use all available logical CPU cores to parallelize threads for executing a single query. Setting MAXDOP to one (1) denotes that only one core can be used for a single query execution. In practical terms, this means that parallelism is turned off. Depending on the case-per-case basis, available cores to the database, and diagnostics log information, you can tune the MAXDOP option to the number of cores used for parallel query execution that might resolve the issue in your case.

Pagelatch Contention

What is happening

This performance pattern indicates the current database workload performance degradation due to pagelatch contention compared to the past seven-day workload baseline.

Latches are lightweight synchronization mechanisms used by SQL Database to enable multithreading. They guarantee consistency of in-memory structures that include indices, data pages, and other internal structures.

There are many types of latches available on the SQL database. For simplicity purposes, buffer latches are used to protect in-memory pages in the buffer pool. IO latches are used to protect pages not yet loaded into the buffer pool. Whenever data is written to or read from a page in the buffer pool, a worker thread needs to acquire a buffer latch for the page first. Whenever a worker thread attempts to access a page that isn't already available in the in-memory buffer pool, an IO request is made to load the required information from the storage. This sequence of events indicates a more severe form of performance degradation.

Contention on the page latches occurs when multiple threads concurrently attempt to acquire latches on the same in-memory structure, which introduces an increased wait time to query execution. In the case of pagelatch IO contention, when data needs to be accessed from storage, this wait time is even larger. It can affect workload performance considerably. Pagelatch contention is the most common scenario of threads waiting on each other and competing for resources on multiple CPU systems.

Troubleshooting

The diagnostics log outputs pagelatch contention details. You can use this information as the basis for troubleshooting.

Because a pagelatch is an internal control mechanism of SQL Database, it automatically determines when to use them. Application decisions, including schema design, can affect pagelatch behavior due to the deterministic behavior of latches.

One method for handling latch contention is to replace a sequential index key with a nonsequential key to evenly distribute inserts across an index range. Typically, a leading column in the index distributes the workload proportionally. Another method to consider is table partitioning. Creating a hash partitioning scheme with a computed column on a partitioned table is a common approach for mitigating excessive latch contention. In the case of pagelatch IO contention, introducing indexes helps to mitigate this performance issue.

For more information, see [Diagnose and resolve latch contention on SQL Server](#) (PDF download).

Missing Index

What is happening

This performance pattern indicates the current database workload performance degradation compared to the past seven-day baseline due to a missing index.

An index is used to speed up the performance of queries. It provides quick access to table data by reducing the number of dataset pages that need to be visited or scanned.

Specific queries that caused performance degradation are identified through this detection for which creating indexes would be beneficial to the performance.

Troubleshooting

The diagnostics log outputs query hashes for the queries that were identified to affect the workload performance. You can build indexes for these queries. You also can optimize or remove these queries if they aren't required. A good performance practice is to avoid querying data that you don't use.

TIP

Did you know that SQL Database built-in intelligence can automatically manage the best-performing indexes for your databases?

For continuous performance optimization of SQL Database, we recommend that you enable [SQL Database automatic tuning](#). This unique feature of SQL Database built-in intelligence continuously monitors your SQL database and automatically tunes and creates indexes for your databases.

New Query

What is happening

This performance pattern indicates that a new query is detected that is performing poorly and affecting the workload performance compared to the seven-day performance baseline.

Writing a good-performing query sometimes can be a challenging task. For more information on writing queries, see [Writing SQL queries](#). To optimize existing query performance, see [Query tuning](#).

Troubleshooting

The diagnostics log outputs information up to two new most CPU-consuming queries, including their query hashes. Because the detected query affects the workload performance, you can optimize your query. Good practice is to retrieve only data you need to use. We also recommend using queries with a WHERE clause. We also recommend that you simplify complex queries and break them up into smaller queries. Another good practice is to break down large batch queries into smaller batch queries. Introducing indexes for new queries is typically a good practice to mitigate this performance issue.

Consider using [Azure SQL Database Query Performance Insight](#).

Increased Wait Statistic

What is happening

This detectable performance pattern indicates a workload performance degradation in which poor-performing queries are identified compared to the past seven-day workload baseline.

In this case, the system can't classify the poor-performing queries under any other standard detectable performance categories, but it detected the wait statistic responsible for the regression. Therefore, it considers them as queries with *increased wait statistics*, where the wait statistic responsible for the regression is also exposed.

Troubleshooting

The diagnostics log outputs information on increased wait time details and query hashes of the affected queries.

Because the system couldn't successfully identify the root cause for the poor-performing queries, the diagnostics information is a good starting point for manual troubleshooting. You can optimize the performance of these queries. A good practice is to fetch only data you need to use and to simplify and break down complex queries into smaller ones.

For more information on optimizing query performance, see [Query tuning](#).

TempDB Contention

What is happening

This detectable performance pattern indicates a database performance condition in which a bottleneck of threads trying to access tempDB resources exists. (This condition isn't IO related.) The typical scenario for this

performance issue is hundreds of concurrent queries that all create, use, and then drop small tempDB tables. The system detected that the number of concurrent queries using the same tempDB tables increased with sufficient statistical significance to affect database performance compared to the past seven-day performance baseline.

Troubleshooting

The diagnostics log outputs tempDB contention details. You can use the information as the starting point for troubleshooting. There are two things you can pursue to alleviate this kind of contention and increase the throughput of the overall workload: You can stop using the temporary tables. You also can use memory-optimized tables.

For more information, see [Introduction to memory-optimized tables](#).

Elastic pool DTU shortage

What is happening

This detectable performance pattern indicates a degradation in the current database workload performance compared to the past seven-day baseline. It's due to the shortage of available DTUs in the elastic pool of your subscription.

Resources on SQL Database are typically referred to as [DTU resources](#), which consist of a blended measure of CPU and IO (data and transaction log IO) resources. [Azure elastic pool resources](#) are used as a pool of available eDTU resources shared between multiple databases for scaling purposes. When available eDTU resources in your elastic pool aren't sufficiently large to support all the databases in the pool, an elastic pool DTU shortage performance issue is detected by the system.

Troubleshooting

The diagnostics log outputs information on the elastic pool, lists the top DTU-consuming databases, and provides a percentage of the pool's DTU used by the top-consuming database.

Because this performance condition is related to multiple databases using the same pool of eDTUs in the elastic pool, the troubleshooting steps focus on the top DTU-consuming databases. You can reduce the workload on the top-consuming databases, which includes optimization of the top-consuming queries on those databases. You also can ensure that you aren't querying data that you don't use. Another approach is to optimize applications by using the top DTU-consuming databases and redistribute the workload among multiple databases.

If reduction and optimization of the current workload on your top DTU-consuming databases aren't possible, consider increasing your elastic pool pricing tier. Such increase results in the increase of the available DTUs in the elastic pool.

Plan Regression

What is happening

This detectable performance pattern denotes a condition in which SQL Database utilizes a suboptimal query execution plan. The suboptimal plan typically causes increased query execution, which leads to longer wait times for the current and other queries.

The SQL database determines the query execution plan with the least cost to a query execution. As the type of queries and workloads change, sometimes the existing plans are no longer efficient, or perhaps SQL Database didn't make a good assessment. As a matter of correction, query execution plans can be manually forced.

This detectable performance pattern combines three different cases of plan regression: new plan regression, old plan regression, and existing plans changed workload. The particular type of plan regression that occurred is provided in the *details* property in the diagnostics log.

The new plan regression condition refers to a state in which SQL Database starts executing a new query

execution plan that isn't as efficient as the old plan. The old plan regression condition refers to the state when SQL Database switches from using a new, more efficient plan to the old plan, which isn't as efficient as the new plan. The existing plans changed workload regression refers to the state in which the old and the new plans continuously alternate, with the balance going more toward the poor-performing plan.

For more information on plan regressions, see [What is plan regression in SQL Server?](#).

Troubleshooting

The diagnostics log outputs the query hashes, good plan ID, bad plan ID, and query IDs. You can use this information as the basis for troubleshooting.

You can analyze which plan is better performing for your specific queries that you can identify with the query hashes provided. After you determine which plan works better for your queries, you can manually force it.

For more information, see [Learn how SQL Server prevents plan regressions](#).

TIP

Did you know that SQL Database built-in intelligence can automatically manage the best-performing query execution plans for your databases?

For continuous performance optimization of SQL Database, we recommend that you enable [SQL Database automatic tuning](#). This unique feature of SQL Database built-in intelligence continuously monitors your SQL database and automatically tunes and creates best-performing query execution plans for your databases.

Database-SScoped Configuration Value Change

What is happening

This detectable performance pattern indicates a condition in which a change in the database-scoped configuration causes performance regression that is detected compared to the past seven-day database workload behavior. This pattern denotes that a recent change made to the database-scoped configuration doesn't seem to be beneficial to your database performance.

Database-scoped configuration changes can be set for each individual database. This configuration is used on a case-by-case basis to optimize the individual performance of your database. The following options can be configured for each individual database: MAXDOP, LEGACY_CARDINALITY_ESTIMATION, PARAMETER_SNIFFING, QUERY_OPTIMIZER_HOTFIXES, and CLEAR PROCEDURE_CACHE.

Troubleshooting

The diagnostics log outputs database-scoped configuration changes that were made recently that caused performance degradation compared to the previous seven-day workload behavior. You can revert the configuration changes to the previous values. You also can tune value by value until the desired performance level is reached. You can copy database-scope configuration values from a similar database with satisfactory performance. If you're unable to troubleshoot the performance, revert to the default SQL Database default values and attempt to fine-tune starting from this baseline.

For more information on optimizing database-scoped configuration and T-SQL syntax on changing the configuration, see [Alter database-scoped configuration \(Transact-SQL\)](#).

Slow Client

What is happening

This detectable performance pattern indicates a condition in which the client using the SQL database can't consume the output from the database as fast as the database sends the results. Because SQL Database isn't storing results of the executed queries in a buffer, it slows down and waits for the client to consume the

transmitted query outputs before proceeding. This condition also might be related to a network that isn't sufficiently fast enough to transmit outputs from the SQL database to the consuming client.

This condition is generated only if a performance regression is detected compared to the past seven-day database workload behavior. This performance issue is detected only if a statistically significant performance degradation occurs compared to previous performance behavior.

Troubleshooting

This detectable performance pattern indicates a client-side condition. Troubleshooting is required at the client-side application or client-side network. The diagnostics log outputs the query hashes and wait times that seem to be waiting the most for the client to consume them within the past two hours. You can use this information as the basis for troubleshooting.

You can optimize performance of your application for consumption of these queries. You also can consider possible network latency issues. Because the performance degradation issue was based on change in the last seven-day performance baseline, you can investigate whether recent application or network condition changes caused this performance regression event.

Pricing Tier Downgrade

What is happening

This detectable performance pattern indicates a condition in which the pricing tier of your SQL Database subscription was downgraded. Because of reduction of resources (DTUs) available to the database, the system detected a drop in the current database performance compared to the past seven-day baseline.

In addition, there could be a condition in which the pricing tier of your SQL Database subscription was downgraded and then upgraded to a higher tier within a short period of time. Detection of this temporary performance degradation is outputted in the details section of the diagnostics log as a pricing tier downgrade and upgrade.

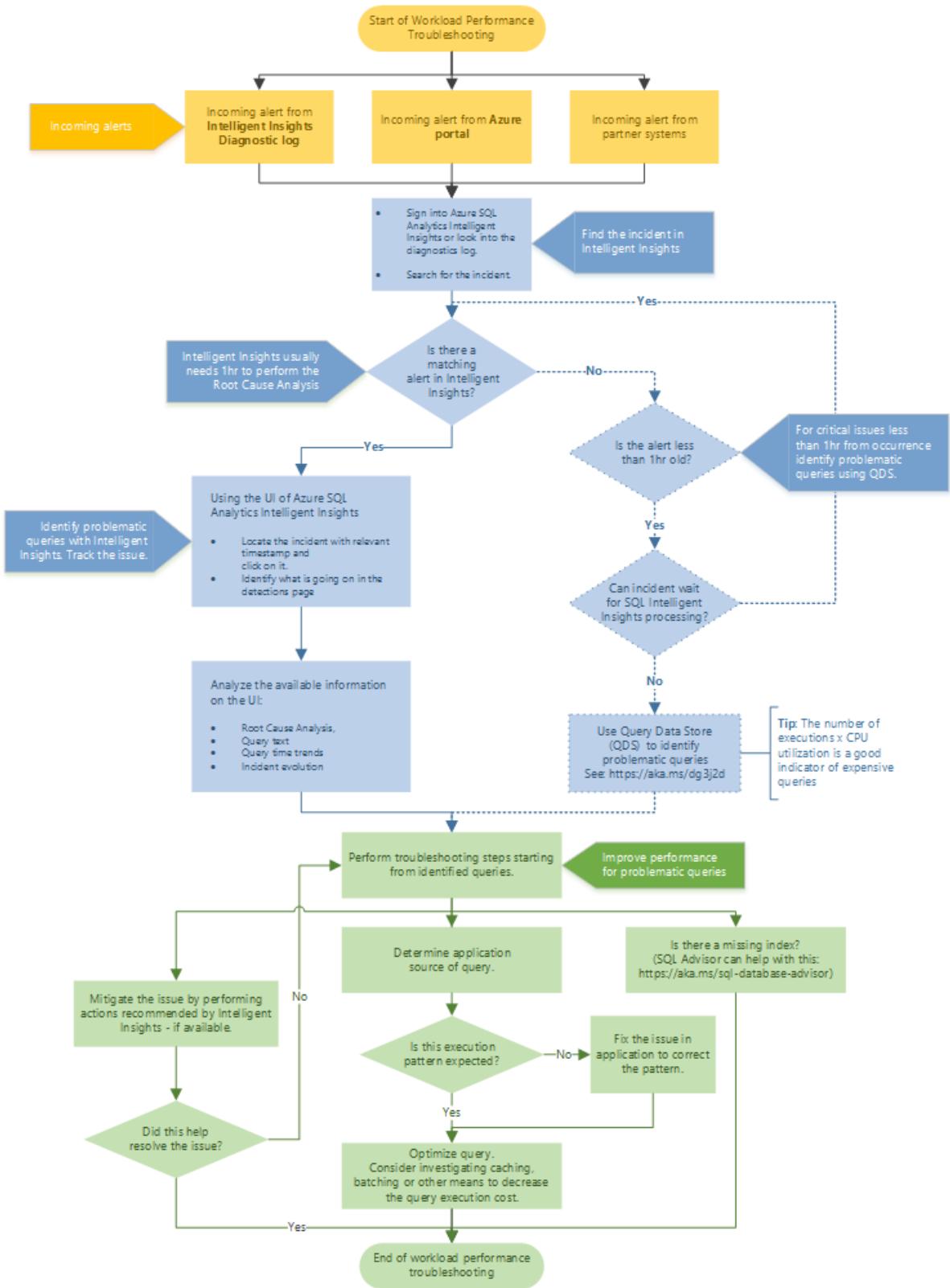
Troubleshooting

If you reduced your pricing tier, and therefore the DTUs available to SQL Database, and you're satisfied with the performance, there's nothing you need to do. If you reduced your pricing tier and you're unsatisfied with your SQL database performance, reduce your database workloads or consider increasing the pricing tier to a higher level.

Recommended troubleshooting flow

Follow the flowchart for a recommended approach to troubleshoot performance issues by using Intelligent Insights.

Access Intelligent Insights through the Azure portal by going to Azure SQL Analytics. Attempt to locate the incoming performance alert, and select it. Identify what is happening on the detections page. Observe the provided root cause analysis of the issue, query text, query time trends, and incident evolution. Attempt to resolve the issue by using the Intelligent Insights recommendation for mitigating the performance issue.



TIP

Select the flowchart to download a PDF version.

Intelligent Insights usually needs one hour of time to perform the root cause analysis of the performance issue. If you can't locate your issue in Intelligent Insights and it's critical to you, use the Query Store to manually identify the root cause of the performance issue. (Typically, these issues are less than one hour old.) For more information, see [Monitor performance by using the Query Store](#).

Next steps

- Learn [Intelligent Insights](#) concepts.
- Use the [Intelligent Insights Azure SQL Database performance diagnostics log](#).
- Monitor [Azure SQL Database](#) by using Azure SQL Analytics.
- Learn to [collect and consume log data from your Azure resources](#).

Use the Intelligent Insights Azure SQL Database performance diagnostics log

2/24/2020 • 6 minutes to read • [Edit Online](#)

This page provides information on how to use the Azure SQL Database performance diagnostics log generated by [Intelligent Insights](#), its format, and the data it contains for your custom development needs. You can send this diagnostics log to [Azure Monitor logs](#), [Azure Event Hubs](#), [Azure Storage](#), or a third-party solution for custom DevOps alerting and reporting capabilities.

Log header

The diagnostics log uses JSON standard format to output Intelligent Insights findings. The exact category property for accessing an Intelligent Insights log is the fixed value "SQLInsights".

The header of the log is common and consists of the time stamp (TimeGenerated) that shows when an entry was created. It also includes a resource ID (ResourceId) that refers to the particular SQL Database the entry relates to. The category (Category), level (Level), and operation name (OperationName) are fixed properties whose values do not change. They indicate that the log entry is informational and that it comes from Intelligent Insights (SQLInsights).

```
"TimeGenerated" : "2017-9-25 11:00:00", // time stamp of the log entry
"ResourceId" : "database identifier", // value points to a database resource
"Category": "SQLInsights", // fixed property
"Level" : "Informational", // fixed property
"OperationName" : "Insight", // fixed property
```

Issue ID and database affected

The issue identification property (issueId_d) provides a way of uniquely tracking performance issues until resolved. Multiple event records in the log reporting status of the same issue will share the same issue ID.

Along with the issue ID, the diagnostics log reports the start (intervalStartTime_t) and end (intervalEndTme_t) time stamps of the particular event related to an issue that's reported in the diagnostics log.

The elastic pool (elasticPoolName_s) property indicates which elastic pool the database with an issue belongs to. If the database isn't part of an elastic pool, this property has no value. The database in which an issue was detected is disclosed in the database name (databaseName_s) property.

```
"intervalStartTime_t": "2017-9-25 11:00", // start of the issue reported time stamp
"intervalEndTme_t": "2017-9-25 12:00", // end of the issue reported time stamp
"elasticPoolName_s" : "", // resource elastic pool (if applicable)
"databaseName_s" : "db_name", // database name
"issueId_d" : 1525, // unique ID of the issue detected
"status_s" : "Active" // status of the issue - possible values: "Active", "Verifying", and "Complete"
```

Detected issues

The next section of the Intelligent Insights performance log contains performance issues that were detected through built-in artificial intelligence. Detections are disclosed in properties within the JSON diagnostics log. These detections consist of the category of an issue, the impact of the issue, the queries affected, and the metrics.

The detections properties might contain multiple performance issues that were detected.

Detected performance issues are reported with the following detections property structure:

```
"detections_s" : [{}  
  "impact" : 1 to 3, // impact of the issue detected, possible values 1-3 (1 low, 2 moderate, 3 high impact)  
  "category" : "Detectable performance pattern", // performance issue detected, see the table  
  "details": <Details outputted> // details of an issue (see the table)  
]
```

Detectable performance patterns and the details that are outputted to the diagnostics log are provided in the following table.

Detection category

The category (category) property describes the category of detectable performance patterns. See the following table for all possible categories of detectable performance patterns. For more information, see [Troubleshoot database performance issues with Intelligent Insights](#).

Depending on the performance issue detected, the details outputted in the diagnostics log file differ accordingly.

| DETECTABLE PERFORMANCE PATTERNS | DETAILS OUTPUTTED |
|---------------------------------|--|
| Reaching resource limits | <ul style="list-style-type: none">• Resources affected• Query hashes• Resource consumption percentage |
| Workload Increase | <ul style="list-style-type: none">• Number of queries whose execution increased• Query hashes of queries with the largest contribution to the workload increase |
| Memory Pressure | <ul style="list-style-type: none">• Memory clerk |
| Locking | <ul style="list-style-type: none">• Affected query hashes• Blocking query hashes |
| Increased MAXDOP | <ul style="list-style-type: none">• Query hashes• CXP wait times• Wait times |
| Pagelatch Contention | <ul style="list-style-type: none">• Query hashes of queries causing contention |
| Missing Index | <ul style="list-style-type: none">• Query hashes |
| New Query | <ul style="list-style-type: none">• Query hash of the new queries |
| Unusual Wait Statistic | <ul style="list-style-type: none">• Unusual wait types• Query hashes• Query wait times |
| TempDB Contention | <ul style="list-style-type: none">• Query hashes of queries causing contention• Query attribution to the overall database pagelatch contention wait time [%] |
| Elastic pool DTU Shortage | <ul style="list-style-type: none">• Elastic pool• Top DTU-consuming database• Percent of pool DTU used by the top consumer |

| DETECTABLE PERFORMANCE PATTERNS | DETAILS OUTPUTTED |
|--|--|
| Plan Regression | <ul style="list-style-type: none"> • Query hashes • Good plan IDs • Bad plan IDs |
| Database-Spaced Configuration Value Change | <ul style="list-style-type: none"> • Database-scoped configuration changes compared to the default values |
| Slow Client | <ul style="list-style-type: none"> • Query hashes • Wait times |
| Pricing Tier Downgrade | <ul style="list-style-type: none"> • Text notification |

Impact

The impact (impact) property describes how much a detected behavior contributed to the problem that a database is having. Impacts range from 1 to 3, with 3 as the highest contribution, 2 as moderate, and 1 as the lowest contribution. The impact value might be used as an input for custom alerting automation, depending on your specific needs. The property queries impacted (QueryHashes) provide a list of the query hashes that were affected by a particular detection.

Impacted queries

The next section of the Intelligent Insights log provides information about particular queries that were affected by the detected performance issues. This information is disclosed as an array of objects embedded in the impact_s property. The impact property consists of entities and metrics. Entities refer to a particular query (Type: Query). The unique query hash is disclosed under the value (Value) property. In addition, each of the queries disclosed is followed by a metric and a value, which indicate a detected performance issue.

In the following log example, the query with the hash 0x9102EXZ4 was detected to have an increased duration of execution (Metric: DurationIncreaseSeconds). The value of 110 seconds indicates that this particular query took 110 seconds longer to execute. Because multiple queries can be detected, this particular log section might include multiple query entries.

```

"impact" : [
  "entity" : {
    "Type" : "Query", // type of entity - query
    "Value" : "query hash value", // for example "0x9102EXZ4" query hash value },
    "Metric" : "DurationIncreaseSeconds", // measured metric and the measurement unit (in this case seconds)
    "Value" : 110 // value of the measured metric (in this case seconds)
  }
]

```

Metrics

The unit of measurement for each metric reported is provided under the metric (metric) property with the possible values of seconds, number, and percentage. The value of a measured metric is reported in the value (value) property.

The DurationIncreaseSeconds property provides the unit of measurement in seconds. The CriticalErrorCount unit of measurement is a number that represents an error count.

```

"metric" : "DurationIncreaseSeconds", // issue metric type - possible values: DurationIncreaseSeconds,
CriticalErrorCount, WaitingSeconds
"value" : 102 // value of the measured metric (in this case seconds)

```

Root cause analysis and improvement recommendations

The last part of the Intelligent Insights performance log pertains to the automated root cause analysis of the identified performance degradation issue. The information appears in human-friendly verbiage in the root cause analysis (`rootCauseAnalysis_s`) property. Improvement recommendations are included in the log where possible.

```
// example of reported root cause analysis of the detected performance issue, in a human-readable format  
  
"rootCauseAnalysis_s" : "High data IO caused performance to degrade. It seems that this database is missing some indexes that could help."
```

You can use the Intelligent Insights performance log with [Azure Monitor logs](#) or a third-party solution for custom DevOps alerting and reporting capabilities.

Next steps

- Learn about [Intelligent Insights](#) concepts.
- Learn how to [troubleshoot Azure SQL Database performance issues with Intelligent Insights](#).
- Learn how to [monitor Azure SQL Database by using Azure SQL Analytics](#).
- Learn how to [collect and consume log data from your Azure resources](#).

Monitor In-Memory OLTP storage

11/6/2019 • 2 minutes to read • [Edit Online](#)

When using [In-Memory OLTP](#), data in memory-optimized tables and table variables resides in In-Memory OLTP storage. Each Premium and Business Critical service tier has a maximum In-Memory OLTP storage size. See [DTU-based resource limits - single database](#), [DTU-based resource limits - elastic pools](#), [vCore-based resource limits - single databases](#) and [vCore-based resource limits - elastic pools](#).

Once this limit is exceeded, insert and update operations may start failing with error 41823 for single databases and error 41840 for elastic pools. At that point you need to either delete data to reclaim memory, or upgrade the service tier or compute size of your database.

Determine whether data fits within the In-Memory OLTP storage cap

Determine the storage caps of the different service tiers. See [DTU-based resource limits - single database](#), [DTU-based resource limits - elastic pools](#), [vCore-based resource limits - single databases](#) and [vCore-based resource limits - elastic pools](#).

Estimating memory requirements for a memory-optimized table works the same way for SQL Server as it does in Azure SQL Database. Take a few minutes to review that article on [MSDN](#).

Table and table variable rows, as well as indexes, count toward the max user data size. In addition, ALTER TABLE needs enough room to create a new version of the entire table and its indexes.

Monitoring and alerting

You can monitor In-memory storage use as a percentage of the storage cap for your compute size in the [Azure portal](#):

1. On the Database blade, locate the Resource utilization box and click on Edit.
2. Select the metric [In-Memory OLTP Storage percentage](#).
3. To add an alert, click on the Resource Utilization box to open the Metric blade, then click on Add alert.

Or use the following query to show the In-memory storage utilization:

```
SELECT xtp_storage_percent FROM sys.dm_db_resource_stats
```

Correct out-of-In-Memory OLTP storage situations - Errors 41823 and 41840

Hitting the In-Memory OLTP storage cap in your database results in INSERT, UPDATE, ALTER and CREATE operations failing with error message 41823 (for single databases) or error 41840 (for elastic pools). Both errors cause the active transaction to abort.

Error messages 41823 and 41840 indicate that the memory-optimized tables and table variables in the database or pool have reached the maximum In-Memory OLTP storage size.

To resolve this error, either:

- Delete data from the memory-optimized tables, potentially offloading the data to traditional, disk-based tables; or,

- Upgrade the service tier to one with enough in-memory storage for the data you need to keep in memory-optimized tables.

NOTE

In rare cases, errors 41823 and 41840 can be transient, meaning there is enough available In-Memory OLTP storage, and retrying the operation succeeds. We therefore recommend to both monitor the overall available In-Memory OLTP storage and to retry when first encountering error 41823 or 41840. For more information about retry logic, see [Conflict Detection and Retry Logic with In-Memory OLTP](#).

Next steps

For monitoring guidance, see [Monitoring Azure SQL Database using dynamic management views](#).

Extended events in SQL Database

11/7/2019 • 5 minutes to read • [Edit Online](#)

This topic explains how the implementation of extended events in Azure SQL Database is slightly different compared to extended events in Microsoft SQL Server.

- SQL Database V12 gained the extended events feature in the second half of calendar 2015.
- SQL Server has had extended events since 2008.
- The feature set of extended events on SQL Database is a robust subset of the features on SQL Server.

XEvents is an informal nickname that is sometimes used for 'extended events' in blogs and other informal locations.

Additional information about extended events, for Azure SQL Database and Microsoft SQL Server, is available at:

- [Quick Start: Extended events in SQL Server](#)
- [Extended Events](#)

Prerequisites

This topic assumes you already have some knowledge of:

- [Azure SQL Database service](#).
- [Extended events](#) in Microsoft SQL Server.
- The bulk of our documentation about extended events applies to both SQL Server and SQL Database.

Prior exposure to the following items is helpful when choosing the Event File as the [target](#):

- [Azure Storage service](#)
- PowerShell
 - [Using Azure PowerShell with Azure Storage](#) - Provides comprehensive information about PowerShell and the Azure Storage service.

Code samples

Related topics provide two code samples:

- [Ring Buffer target code for extended events in SQL Database](#)
 - Short simple Transact-SQL script.
 - We emphasize in the code sample topic that, when you are done with a Ring Buffer target, you should release its resources by executing an alter-drop `ALTER EVENT SESSION ... ON DATABASE DROP TARGET ...;` statement. Later you can add another instance of Ring Buffer by `ALTER EVENT SESSION ... ON DATABASE ADD TARGET ...;`
- [Event File target code for extended events in SQL Database](#)
 - Phase 1 is PowerShell to create an Azure Storage container.
 - Phase 2 is Transact-SQL that uses the Azure Storage container.

Transact-SQL differences

- When you execute the [CREATE EVENT SESSION](#) command on SQL Server, you use the **ON SERVER** clause. But on SQL Database you use the **ON DATABASE** clause instead.
- The **ON DATABASE** clause also applies to the [ALTER EVENT SESSION](#) and [DROP EVENT SESSION](#) Transact-SQL commands.
- A best practice is to include the event session option of **STARTUP_STATE = ON** in your [CREATE EVENT SESSION](#) or [ALTER EVENT SESSION](#) statements.
 - The **= ON** value supports an automatic restart after a reconfiguration of the logical database due to a failover.

New catalog views

The extended events feature is supported by several [catalog views](#). Catalog views tell you about *metadata or definitions* of user-created event sessions in the current database. The views do not return information about instances of active event sessions.

| NAME OF CATALOG VIEW | DESCRIPTION |
|---|---|
| sys.database_event_session_actions | Returns a row for each action on each event of an event session. |
| sys.database_event_session_events | Returns a row for each event in an event session. |
| sys.database_event_session_fields | Returns a row for each customize-able column that was explicitly set on events and targets. |
| sys.database_event_session_targets | Returns a row for each event target for an event session. |
| sys.database_event_sessions | Returns a row for each event session in the SQL Database database. |

In Microsoft SQL Server, similar catalog views have names that include `.server_` instead of `.database_`. The name pattern is like **sys.server_event_%**.

New dynamic management views (DMVs)

Azure SQL Database has [dynamic management views \(DMVs\)](#) that support extended events. DMVs tell you about *active* event sessions.

| NAME OF DMV | DESCRIPTION |
|--|---|
| sys.dm_xe_database_session_event_actions | Returns information about event session actions. |
| sys.dm_xe_database_session_events | Returns information about session events. |
| sys.dm_xe_database_session_object_columns | Shows the configuration values for objects that are bound to a session. |
| sys.dm_xe_database_session_targets | Returns information about session targets. |

| NAME OF DMV | DESCRIPTION |
|------------------------------------|--|
| sys.dm_xe_database_sessions | Returns a row for each event session that is scoped to the current database. |

In Microsoft SQL Server, similar catalog views are named without the *_database* portion of the name, such as:

- **sys.dm_xe_sessions**, instead of name **sys.dm_xe_database_sessions**.

DMVs common to both

For extended events there are additional DMVs that are common to both Azure SQL Database and Microsoft SQL Server:

- **sys.dm_xe_map_values**
- **sys.dm_xe_object_columns**
- **sys.dm_xe_objects**
- **sys.dm_xe_packages**

Find the available extended events, actions, and targets

You can run a simple SQL **SELECT** to obtain a list of the available events, actions, and target.

```

SELECT
    o.object_type,
    p.name      AS [package_name],
    o.name       AS [db_object_name],
    o.description AS [db_obj_description]
FROM
    sys.dm_xe_objects  AS o
INNER JOIN sys.dm_xe_packages AS p  ON p.guid = o.package_guid
WHERE
    o.object_type IN
    (
        'action', 'event', 'target'
    )
ORDER BY
    o.object_type,
    p.name,
    o.name;

```

Targets for your SQL Database event sessions

Here are targets that can capture results from your event sessions on SQL Database:

- [Ring Buffer target](#) - Briefly holds event data in memory.
- [Event Counter target](#) - Counts all events that occur during an extended events session.
- [Event File target](#) - Writes complete buffers to an Azure Storage container.

The [Event Tracing for Windows \(ETW\)](#) API is not available for extended events on SQL Database.

Restrictions

There are a couple of security-related differences befitting the cloud environment of SQL Database:

- Extended events are founded on the single-tenant isolation model. An event session in one database cannot access data or events from another database.
- You cannot issue a **CREATE EVENT SESSION** statement in the context of the **master** database.

Permission model

You must have **Control** permission on the database to issue a **CREATE EVENT SESSION** statement. The database owner (dbo) has **Control** permission.

Storage container authorizations

The SAS token you generate for your Azure Storage container must specify **rwl** for the permissions. The **rwl** value provides the following permissions:

- Read
- Write
- List

Performance considerations

There are scenarios where intensive use of extended events can accumulate more active memory than is healthy for the overall system. Therefore the Azure SQL Database system dynamically sets and adjusts limits on the amount of active memory that can be accumulated by an event session. Many factors go into the dynamic calculation.

If you receive an error message that says a memory maximum was enforced, some corrective actions you can take are:

- Run fewer concurrent event sessions.
- Through your **CREATE** and **ALTER** statements for event sessions, reduce the amount of memory you specify on the **MAX_MEMORY** clause.

Network latency

The **Event File** target might experience network latency or failures while persisting data to Azure Storage blobs. Other events in SQL Database might be delayed while they wait for the network communication to complete. This delay can slow your workload.

- To mitigate this performance risk, avoid setting the **EVENT_RETENTION_MODE** option to **NO_EVENT_LOSS** in your event session definitions.

Related links

- [Using Azure PowerShell with Azure Storage](#).
- [Azure Storage Cmdlets](#)
- [Using Azure PowerShell with Azure Storage](#) - Provides comprehensive information about PowerShell and the Azure Storage service.
- [How to use Blob storage from .NET](#)
- [CREATE CREDENTIAL \(Transact-SQL\)](#)
- [CREATE EVENT SESSION \(Transact-SQL\)](#)
- [Jonathan Kehayias' blog posts about extended events in Microsoft SQL Server](#)
- The Azure *Service Updates* webpage, narrowed by parameter to Azure SQL Database:

- o <https://azure.microsoft.com/updates/?service=sql-database>

Other code sample topics for extended events are available at the following links. However, you must routinely check any sample to see whether the sample targets Microsoft SQL Server versus Azure SQL Database. Then you can decide whether minor changes are needed to run the sample.

Event File target code for extended events in SQL Database

11/22/2019 • 10 minutes to read • [Edit Online](#)

You want a complete code sample for a robust way to capture and report information for an extended event.

In Microsoft SQL Server, the [Event File target](#) is used to store event outputs into a local hard drive file. But such files are not available to Azure SQL Database. Instead we use the Azure Storage service to support the Event File target.

This topic presents a two-phase code sample:

- PowerShell, to create an Azure Storage container in the cloud.
- Transact-SQL:
 - To assign the Azure Storage container to an Event File target.
 - To create and start the event session, and so on.

Prerequisites

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

- An Azure account and subscription. You can sign up for a [free trial](#).
- Any database you can create a table in.
 - Optionally you can [create an AdventureWorksLT demonstration database](#) in minutes.
- SQL Server Management Studio (ssms.exe), ideally its latest monthly update version. You can download the latest ssms.exe from:
 - Topic titled [Download SQL Server Management Studio](#).
 - [A direct link to the download](#).
- You must have the [Azure PowerShell modules](#) installed.
 - The modules provide commands such as - **New-AzStorageAccount**.

Phase 1: PowerShell code for Azure Storage container

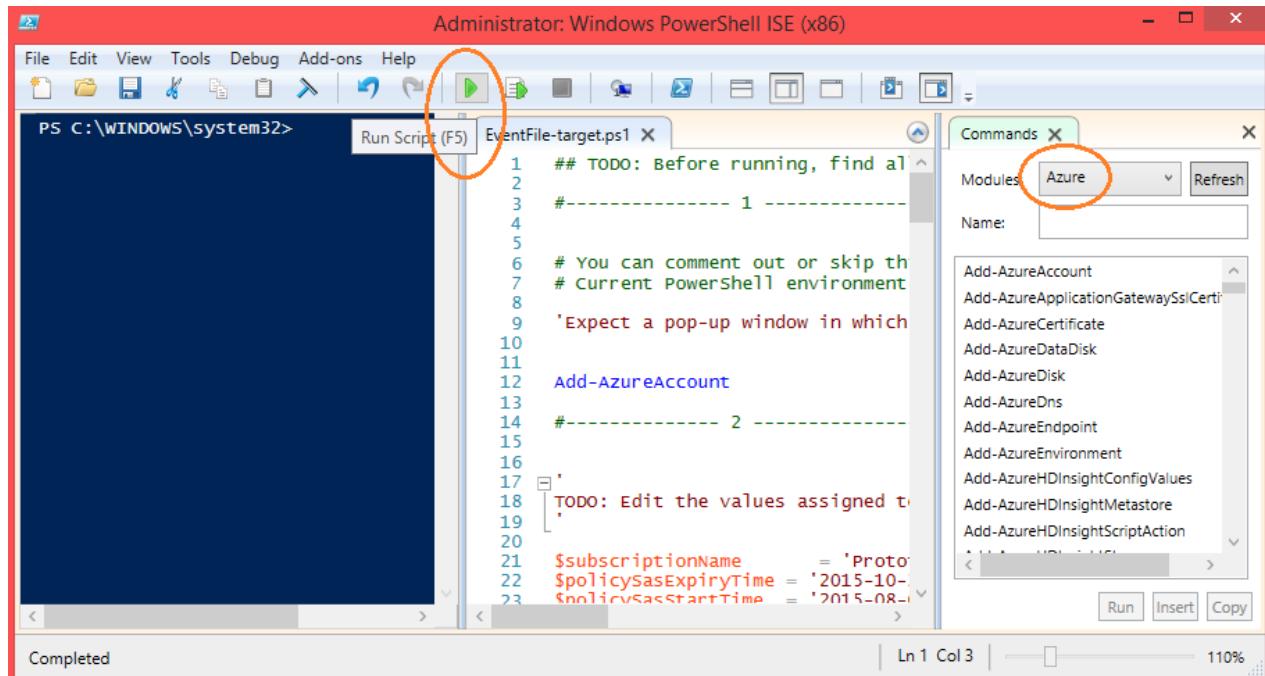
This PowerShell is phase 1 of the two-phase code sample.

The script starts with commands to clean up after a possible previous run, and is rerunnable.

1. Paste the PowerShell script into a simple text editor such as Notepad.exe, and save the script as a file with the extension **.ps1**.
2. Start PowerShell ISE as an Administrator.
3. At the prompt, type

```
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser
```

and then press Enter.
4. In PowerShell ISE, open your **.ps1** file. Run the script.
5. The script first starts a new window in which you log in to Azure.
 - If you rerun the script without disrupting your session, you have the convenient option of commenting out the **Add-AzureAccount** command.



PowerShell code

This PowerShell script assumes you have already installed the Az module. For information, see [Install the Azure PowerShell module](#).

```
## TODO: Before running, find all 'TODO' and make each edit!!

cls;

#----- 1 -----


'Script assumes you have already logged your PowerShell session into Azure.
But if not, run Connect-AzAccount (or Connect-AzAccount), just one time.';
#Connect-AzAccount; # Same as Connect-AzAccount.

#----- 2 -----


'

TODO: Edit the values assigned to these variables, especially the first few!
';

# Ensure the current date is between
# the Expiry and Start time values that you edit here.

$subscriptionName      = 'YOUR SUBSCRIPTION NAME';
```

```

$resourceGroupName    = 'YOUR_RESOURCE-GROUP-NAME';

$policySasExpiryTime = '2018-08-28T23:44:56Z';
$policySasStartTime  = '2017-10-01';

$storageAccountLocation = 'YOUR_STORAGE_ACCOUNT_LOCATION';
$storageAccountName     = 'YOUR_STORAGE_ACCOUNT_NAME';
$contextName            = 'YOUR_CONTEXT_NAME';
$containerName          = 'YOUR_CONTAINER_NAME';
$policySasToken         = ' ? ';

$policySasPermission = 'rwl'; # Leave this value alone, as 'rwl'.

#----- 3 -----

# The ending display lists your Azure subscriptions.
# One should match the $subscriptionName value you assigned
# earlier in this PowerShell script.

'Choose an existing subscription for the current PowerShell environment.';

Select-AzSubscription -Subscription $subscriptionName;

#----- 4 -----


'
Clean up the old Azure Storage Account after any previous run,
before continuing this new run.';

if ($storageAccountName) {
    Remove-AzStorageAccount `

        -Name           $storageAccountName `

        -ResourceGroupName $resourceGroupName;
}

#----- 5 -----


[System.DateTime]::Now.ToString();

'

Create a storage account.
This might take several minutes, will beep when ready.
...PLEASE WAIT...';

New-AzStorageAccount `

    -Name           $storageAccountName `

    -Location       $storageAccountLocation `

    -ResourceGroupName $resourceGroupName `

    -SkuName        'Standard_LRS';

[System.DateTime]::Now.ToString();
[System.Media.SystemSounds]::Beep.Play();

'

Get the access key for your storage account.
';

$accessKey_ForStorageAccount = `

    (Get-AzStorageAccountKey `

        -Name           $storageAccountName `

        -ResourceGroupName $resourceGroupName

    ).Value[0];

````$accessKey_ForStorageAccount = $accessKey_ForStorageAccount````

'Azure Storage Account cmdlet completed.
Remainder of PowerShell .ps1 script continues.
';

```

```

----- 6 -----

The context will be needed to create a container within the storage account.

'Create a context object from the storage account and its primary access key.
';

$context = New-AzStorageContext `

 -StorageAccountName $storageAccountName `

 -StorageAccountKey $accessKey_ForStorageAccount;

'Create a container within the storage account.
';

$containerObjectInStorageAccount = New-AzStorageContainer `

 -Name $containerName `

 -Context $context;

'Create a security policy to be applied to the SAS token.
';

New-AzStorageContainerStoredAccessPolicy `

 -Container $containerName `

 -Context $context `

 -Policy $policySasToken `

 -Permission $policySasPermission `

 -ExpiryTime $policySasExpiryTime `

 -StartTime $policySasStartTime;

'

Generate a SAS token for the container.
';

try {
 $sasTokenWithPolicy = New-AzStorageContainerSASToken `

 -Name $containerName `

 -Context $context `

 -Policy $policySasToken;
}

catch {
 $Error[0].Exception.ToString();
}

#----- 7 -----

'Display the values that YOU must edit into the Transact-SQL script next!:
';

"storageAccountName: $storageAccountName";
"containerName: $containerName";
"sasTokenWithPolicy: $sasTokenWithPolicy";

'

REMINDER: sasTokenWithPolicy here might start with "?" character, which you must exclude from Transact-SQL.
';

'

(Later, return here to delete your Azure Storage account. See the preceding Remove-AzStorageAccount -Name $storageAccountName');

'

Now shift to the Transact-SQL portion of the two-part code sample!';

EOFfile

```

Take note of the few named values that the PowerShell script prints when it ends. You must edit those values into the Transact-SQL script that follows as phase 2.

## Phase 2: Transact-SQL code that uses Azure Storage container

- In phase 1 of this code sample, you ran a PowerShell script to create an Azure Storage container.
- Next in phase 2, the following Transact-SQL script must use the container.

The script starts with commands to clean up after a possible previous run, and is rerunnable.

The PowerShell script printed a few named values when it ended. You must edit the Transact-SQL script to use those values. Find **TODO** in the Transact-SQL script to locate the edit points.

1. Open SQL Server Management Studio (ssms.exe).
2. Connect to your Azure SQL Database database.
3. Click to open a new query pane.
4. Paste the following Transact-SQL script into the query pane.
5. Find every **TODO** in the script and make the appropriate edits.
6. Save, and then run the script.

### WARNING

The SAS key value generated by the preceding PowerShell script might begin with a '?' (question mark). When you use the SAS key in the following T-SQL script, you must *remove the leading '?'*. Otherwise your efforts might be blocked by security.

### Transact-SQL code

```
---- TODO: First, run the earlier PowerShell portion of this two-part code sample.
---- TODO: Second, find every 'TODO' in this Transact-SQL file, and edit each.

---- Transact-SQL code for Event File target on Azure SQL Database.

SET NOCOUNT ON;
GO

---- Step 1. Establish one little table, and -----
---- insert one row of data.

IF EXISTS
 (SELECT * FROM sys.objects
 WHERE type = 'U' and name = 'gmTabEmployee')
BEGIN
 DROP TABLE gmTabEmployee;
END
GO

CREATE TABLE gmTabEmployee
(
 EmployeeGuid uniqueIdentifier not null default newid() primary key,
 EmployeeId int not null identity(1,1),
 EmployeeKudosCount int not null default 0,
 EmployeeDescr nvarchar(256) null
);
GO

INSERT INTO gmTabEmployee (EmployeeDescr)
 VALUES ('Jane Doe');
GO

---- Step 2. Create key, and -----
---- Create credential (your Azure Storage container must already exist).

IF NOT EXISTS
 (SELECT * FROM sys.symmetric_keys
 WHERE symmetric_key_id = 101)
```

```

BEGIN
 CREATE MASTER KEY ENCRYPTION BY PASSWORD = '0C34C960-6621-4682-A123-C7EA08E3FC46' -- Or any newid().
END
GO

IF EXISTS
 (SELECT * FROM sys.database_scoped_credentials
 -- TODO: Assign AzureStorageAccount name, and the associated Container name.
 WHERE name = 'https://gmstorageaccountxevent.blob.core.windows.net/gmcontainerxevent')
BEGIN
 DROP DATABASE SCOPED CREDENTIAL
 -- TODO: Assign AzureStorageAccount name, and the associated Container name.
 [https://gmstorageaccountxevent.blob.core.windows.net/gmcontainerxevent] ;
END
GO

CREATE
 DATABASE SCOPED
 CREDENTIAL
 -- use '.blob.', and not '.queue.' or '.table.' etc.
 -- TODO: Assign AzureStorageAccount name, and the associated Container name.
 [https://gmstorageaccountxevent.blob.core.windows.net/gmcontainerxevent]
 WITH
 IDENTITY = 'SHARED ACCESS SIGNATURE', -- "SAS" token.
 -- TODO: Paste in the long SasToken string here for Secret, but exclude any leading '?'.
 SECRET = 'sv=2014-02-14&sr=c&si=gmpolicysastoken&sig=EjAqjo6Nu5xMLEZEkMkLbeF7TD9v1J8DNB2t8g0KTts%3D'
 ;
GO

----- Step 3. Create (define) an event session. -----
----- The event session has an event with an action,
----- and a has a target.

IF EXISTS
 (SELECT * from sys.database_event_sessions
 WHERE name = 'gmeventsessionname240b')
BEGIN
 DROP
 EVENT SESSION
 gmeventsessionname240b
 ON DATABASE;
END
GO

CREATE
 EVENT SESSION
 gmeventsessionname240b
 ON DATABASE

 ADD EVENT
 sqlserver.sql_statement_starting
 (
 ACTION (sqlserver.sql_text)
 WHERE statement LIKE 'UPDATE gmTabEmployee%'
)
 ADD TARGET
 package0.event_file
 (
 -- TODO: Assign AzureStorageAccount name, and the associated Container name.
 -- Also, tweak the .xel file name at end, if you like.
 SET filename =
 'https://gmstorageaccountxevent.blob.core.windows.net/gmcontainerxevent/anyfilename.xel'
)
 WITH
 (MAX_MEMORY = 10 MB,
 MAX_DISPATCH_LATENCY = 3 SECONDS)
 ;
GO

```

```

--

----- Step 4. Start the event session. -----

----- Issue the SQL Update statements that will be traced.

----- Then stop the session.

----- Note: If the target fails to attach,

----- the session must be stopped and restarted.

ALTER

 EVENT SESSION

 gmeventsessionsessionname240b

 ON DATABASE

 STATE = START;

GO

SELECT 'BEFORE_Updates', EmployeeKudosCount, * FROM gmTabEmployee;

UPDATE gmTabEmployee

 SET EmployeeKudosCount = EmployeeKudosCount + 2

 WHERE EmployeeDescr = 'Jane Doe';

UPDATE gmTabEmployee

 SET EmployeeKudosCount = EmployeeKudosCount + 13

 WHERE EmployeeDescr = 'Jane Doe';

SELECT 'AFTER__Updates', EmployeeKudosCount, * FROM gmTabEmployee;

GO

ALTER

 EVENT SESSION

 gmeventsessionsessionname240b

 ON DATABASE

 STATE = STOP;

GO

----- Step 5. Select the results. -----

SELECT

 *, 'CLICK_NEXT_CELL_TO_BROWSE_ITS_RESULTS!' as [CLICK_NEXT_CELL_TO_BROWSE_ITS_RESULTS],

 CAST(event_data AS XML) AS [event_data_XML] -- TODO: In ssms.exe results grid, double-click this

cell!

 FROM

 sys.fn_xe_file_target_read_file

 (

 -- TODO: Fill in Storage Account name, and the associated Container name.

 -- TODO: The name of the .xel file needs to be an exact match to the files in the storage

account Container (You can use Storage Account explorer from the portal to find out the exact file names or

you can retrieve the name using the following DMV-query: select target_data from

sys.dm_xe_database_session_targets. The 3rd xml-node, "File name", contains the name of the file currently

written to.)

 'https://gmstorageaccountxe.event.blob.core.windows.net/gmcontainerxeevent/anyfilename.xel242b',

 null, null, null

);

GO

----- Step 6. Clean up. -----

DROP

 EVENT SESSION

 gmeventsessionsessionname240b

 ON DATABASE;

GO

DROP DATABASE SCOPED CREDENTIAL

 -- TODO: Assign AzureStorageAccount name, and the associated Container name.

 [https://gmstorageaccountxe.event.blob.core.windows.net/gmcontainerxeevent]

 ;

GO

```

```

DROP TABLE gmTabEmployee;
GO

PRINT 'Use PowerShell Remove-AzStorageAccount to delete your Azure Storage account!';
GO

```

If the target fails to attach when you run, you must stop and restart the event session:

```

ALTER EVENT SESSION ... STATE = STOP;
GO
ALTER EVENT SESSION ... STATE = START;
GO

```

## Output

When the Transact-SQL script completes, click a cell under the **event\_data\_XML** column header. One **<event>** element is displayed which shows one UPDATE statement.

Here is one **<event>** element that was generated during testing:

```

<event name="sql_statement_starting" package="sqlserver" timestamp="2015-09-22T19:18:45.420Z">
 <data name="state">
 <value>0</value>
 <text>Normal</text>
 </data>
 <data name="line_number">
 <value>5</value>
 </data>
 <data name="offset">
 <value>148</value>
 </data>
 <data name="offset_end">
 <value>368</value>
 </data>
 <data name="statement">
 <value>UPDATE gmTabEmployee
 SET EmployeeKudosCount = EmployeeKudosCount + 2
 WHERE EmployeeDescr = 'Jane Doe'</value>
 </data>
 <action name="sql_text" package="sqlserver">
 <value>
 SELECT 'BEFORE_Updates', EmployeeKudosCount, * FROM gmTabEmployee;

 UPDATE gmTabEmployee
 SET EmployeeKudosCount = EmployeeKudosCount + 2
 WHERE EmployeeDescr = 'Jane Doe';

 UPDATE gmTabEmployee
 SET EmployeeKudosCount = EmployeeKudosCount + 13
 WHERE EmployeeDescr = 'Jane Doe';

 SELECT 'AFTER__Updates', EmployeeKudosCount, * FROM gmTabEmployee;
 </value>
 </action>
</event>

```

The preceding Transact-SQL script used the following system function to read the event\_file:

- [sys.fn\\_xe\\_file\\_target\\_read\\_file \(Transact-SQL\)](#)

An explanation of advanced options for the viewing of data from extended events is available at:

- [Advanced Viewing of Target Data from Extended Events](#)

## Converting the code sample to run on SQL Server

Suppose you wanted to run the preceding Transact-SQL sample on Microsoft SQL Server.

- For simplicity, you would want to completely replace use of the Azure Storage container with a simple file such as `C:\myeventdata.xel`. The file would be written to the local hard drive of the computer that hosts SQL Server.
- You would not need any kind of Transact-SQL statements for **CREATE MASTER KEY** and **CREATE CREDENTIAL**.
- In the **CREATE EVENT SESSION** statement, in its **ADD TARGET** clause, you would replace the `Http` value assigned made to `filename=` with a full path string like `C:\myfile.xel`.
  - No Azure Storage account need be involved.

## More information

For more info about accounts and containers in the Azure Storage service, see:

- [How to use Blob storage from .NET](#)
- [Naming and Referencing Containers, Blobs, and Metadata](#)
- [Working with the Root Container](#)
- [Lesson 1: Create a stored access policy and a shared access signature on an Azure container](#)
  - [Lesson 2: Create a SQL Server credential using a shared access signature](#)
- [Extended Events for Microsoft SQL Server](#)

# Ring Buffer target code for extended events in SQL Database

11/7/2019 • 5 minutes to read • [Edit Online](#)

You want a complete code sample for the easiest quick way to capture and report information for an extended event during a test. The easiest target for extended event data is the [Ring Buffer target](#).

This topic presents a Transact-SQL code sample that:

1. Creates a table with data to demonstrate with.
2. Creates a session for an existing extended event, namely **sqlserver.sql\_statement\_starting**.
  - The event is limited to SQL statements that contain a particular Update string: **statement LIKE '%UPDATE tabEmployee%'**.
  - Chooses to send the output of the event to a target of type Ring Buffer, namely **package0.ring\_buffer**.
3. Starts the event session.
4. Issues a couple of simple SQL UPDATE statements.
5. Issues a SQL SELECT statement to retrieve event output from the Ring Buffer.
  - **sys.dm\_xe\_database\_session\_targets** and other dynamic management views (DMVs) are joined.
6. Stops the event session.
7. Drops the Ring Buffer target, to release its resources.
8. Drops the event session and the demo table.

## Prerequisites

- An Azure account and subscription. You can sign up for a [free trial](#).
- Any database you can create a table in.
  - Optionally you can [create an AdventureWorksLT demonstration database](#) in minutes.
- SQL Server Management Studio (ssms.exe), ideally its latest monthly update version. You can download the latest ssms.exe from:
  - Topic titled [Download SQL Server Management Studio](#).
  - [A direct link to the download](#).

## Code sample

With very minor modification, the following Ring Buffer code sample can be run on either Azure SQL Database or Microsoft SQL Server. The difference is the presence of the node '\_database' in the name of some dynamic management views (DMVs), used in the FROM clause in Step 5. For example:

- **sys.dm\_xe\_database\_session\_targets**
- **sys.dm\_xe\_session\_targets**

```

GO
---- Transact-SQL.
---- Step set 1.

SET NOCOUNT ON;
GO

IF EXISTS
 (SELECT * FROM sys.objects
 WHERE type = 'U' and name = 'tabEmployee')
BEGIN
 DROP TABLE tabEmployee;
END
GO

CREATE TABLE tabEmployee
(
 EmployeeGuid uniqueIdentifier not null default newid() primary key,
 EmployeeId int not null identity(1,1),
 EmployeeKudosCount int not null default 0,
 EmployeeDescr nvarchar(256) null
);
GO

INSERT INTO tabEmployee (EmployeeDescr)
 VALUES ('Jane Doe');
GO

---- Step set 2.

IF EXISTS
 (SELECT * from sys.database_event_sessions
 WHERE name = 'eventsession_gm_azuresqlldb51')
BEGIN
 DROP EVENT SESSION eventsession_gm_azuresqlldb51
 ON DATABASE;
END
GO

CREATE
 EVENT SESSION eventsession_gm_azuresqlldb51
 ON DATABASE
 ADD EVENT
 sqlserver.sql_statement_starting
 (
 ACTION (sqlserver.sql_text)
 WHERE statement LIKE '%UPDATE tabEmployee%'
)
 ADD TARGET
 package0.ring_buffer
 (SET
 max_memory = 500 -- Units of KB.
);
GO

---- Step set 3.

ALTER EVENT SESSION eventsession_gm_azuresqlldb51
 ON DATABASE
 STATE = START;
GO

---- Step set 4.

```

```

SELECT 'BEFORE_Updates', EmployeeKudosCount, * FROM tabEmployee;

UPDATE tabEmployee
 SET EmployeeKudosCount = EmployeeKudosCount + 102;

UPDATE tabEmployee
 SET EmployeeKudosCount = EmployeeKudosCount + 1015;

SELECT 'AFTER__Updates', EmployeeKudosCount, * FROM tabEmployee;
GO

---- Step set 5.

SELECT
 se.name AS [session-name],
 ev.event_name,
 ac.action_name,
 st.target_name,
 se.session_source,
 st.target_data,
 CAST(st.target_data AS XML) AS [target_data_XML]
FROM
 sys.dm_xe_database_session_event_actions AS ac
 INNER JOIN sys.dm_xe_database_session_events AS ev ON ev.event_name = ac.event_name
 AND CAST(ev.event_session_address AS BINARY(8)) = CAST(ac.event_session_address AS BINARY(8))
 INNER JOIN sys.dm_xe_database_session_object_columns AS oc
 ON CAST(oc.event_session_address AS BINARY(8)) = CAST(ac.event_session_address AS BINARY(8))
 INNER JOIN sys.dm_xe_database_session_targets AS st
 ON CAST(st.event_session_address AS BINARY(8)) = CAST(ac.event_session_address AS BINARY(8))
 INNER JOIN sys.dm_xe_database_sessions AS se
 ON CAST(ac.event_session_address AS BINARY(8)) = CAST(se.address AS BINARY(8))
WHERE
 oc.column_name = 'occurrence_number'
 AND
 se.name = 'eventsession_gm_azuresqlldb51'
 AND
 ac.action_name = 'sql_text'
ORDER BY
 se.name,
 ev.event_name,
 ac.action_name,
 st.target_name,
 se.session_source
;
GO

---- Step set 6.

ALTER EVENT SESSION eventsession_gm_azuresqlldb51
 ON DATABASE
 STATE = STOP;
GO

---- Step set 7.

ALTER EVENT SESSION eventsession_gm_azuresqlldb51
 ON DATABASE
 DROP TARGET package0.ring_buffer;
GO

```

```
---- Step set 8.
```

```
DROP EVENT SESSION eventsession_gm_azuresqlldb51
 ON DATABASE;
GO

DROP TABLE tabEmployee;
GO
```

## Ring Buffer contents

We used ssms.exe to run the code sample.

To view the results, we clicked the cell under the column header **target\_data\_XML**.

Then in the results pane we clicked the cell under the column header **target\_data\_XML**. This click created another file tab in ssms.exe in which the content of the result cell was displayed, as XML.

The output is shown in the following block. It looks long, but it is just two **<event>** elements.

```
<RingBufferTarget truncated="0" processingTime="0" totalEventsProcessed="2" eventCount="2" droppedCount="0"
memoryUsed="1728">
<event name="sql_statement_starting" package="sqlserver" timestamp="2015-09-22T15:29:31.317Z">
<data name="state">
<type name="statement_starting_state" package="sqlserver" />
<value>0</value>
<text>Normal</text>
</data>
<data name="line_number">
<type name="int32" package="package0" />
<value>7</value>
</data>
<data name="offset">
<type name="int32" package="package0" />
<value>184</value>
</data>
<data name="offset_end">
<type name="int32" package="package0" />
<value>328</value>
</data>
<data name="statement">
<type name="unicode_string" package="package0" />
<value>UPDATE tabEmployee
SET EmployeeKudosCount = EmployeeKudosCount + 102</value>
</data>
<action name="sql_text" package="sqlserver">
<type name="unicode_string" package="package0" />
<value>
---- Step set 4.
```

```
SELECT 'BEFORE_Updates', EmployeeKudosCount, * FROM tabEmployee;
```

```
UPDATE tabEmployee
SET EmployeeKudosCount = EmployeeKudosCount + 102;
```

```
UPDATE tabEmployee
SET EmployeeKudosCount = EmployeeKudosCount + 1015;
```

```
SELECT 'AFTER_Updates', EmployeeKudosCount, * FROM tabEmployee;
</values>
```

```

</value>
</action>
</event>
<event name="sql_statement_starting" package="sqlserver" timestamp="2015-09-22T15:29:31.327Z">
<data name="state">
<type name="statement_starting_state" package="sqlserver" />
<value>0</value>
<text>Normal</text>
</data>
<data name="line_number">
<type name="int32" package="package0" />
<value>10</value>
</data>
<data name="offset">
<type name="int32" package="package0" />
<value>340</value>
</data>
<data name="offset_end">
<type name="int32" package="package0" />
<value>486</value>
</data>
<data name="statement">
<type name="unicode_string" package="package0" />
<value>UPDATE tabEmployee
SET EmployeeKudosCount = EmployeeKudosCount + 1015</value>
</data>
<action name="sql_text" package="sqlserver">
<type name="unicode_string" package="package0" />
<value>
---- Step set 4.

```

```
SELECT 'BEFORE_Updates', EmployeeKudosCount, * FROM tabEmployee;
```

```
UPDATE tabEmployee
SET EmployeeKudosCount = EmployeeKudosCount + 102;
```

```
UPDATE tabEmployee
SET EmployeeKudosCount = EmployeeKudosCount + 1015;
```

```
SELECT 'AFTER__Updates', EmployeeKudosCount, * FROM tabEmployee;
```

```
</value>
</action>
</event>
</RingBufferTarget>
```

#### **Release resources held by your Ring Buffer**

When you are done with your Ring Buffer, you can remove it and release its resources issuing an **ALTER** like the following:

```
ALTER EVENT SESSION eventsession_gm_azuresqlDb51
ON DATABASE
DROP TARGET package0.ring_buffer;
GO
```

The definition of your event session is updated, but not dropped. Later you can add another instance of the Ring Buffer to your event session:

```
ALTER EVENT SESSION eventsession_gm_azuresqldb51
 ON DATABASE
 ADD TARGET
 package0.ring_buffer
 (
 SET
 max_memory = 500 -- Units of KB.
);

```

## More information

The primary topic for extended events on Azure SQL Database is:

- [Extended event considerations in SQL Database](#), which contrasts some aspects of extended events that differ between Azure SQL Database versus Microsoft SQL Server.

Other code sample topics for extended events are available at the following links. However, you must routinely check any sample to see whether the sample targets Microsoft SQL Server versus Azure SQL Database. Then you can decide whether minor changes are needed to run the sample.

- Code sample for Azure SQL Database: [Event File target code for extended events in SQL Database](#)

# SQL Hyperscale performance troubleshooting diagnostics

1/2/2020 • 7 minutes to read • [Edit Online](#)

To troubleshoot performance problems in a Hyperscale database, [general performance tuning methodologies](#) on the Azure SQL database compute node is the starting point of a performance investigation. However, given the [distributed architecture](#) of Hyperscale, additional diagnostics have been added to assist. This article describes Hyperscale-specific diagnostic data.

## Log rate throttling waits

Every Azure SQL Database service level has log generation rate limits enforced via [log rate governance](#). In Hyperscale, the log generation limit is currently set to 100 MB/sec, regardless of the service level. However, there are times when the log generation rate on the primary compute replica has to be throttled to maintain recoverability SLAs. This throttling happens when a [page server or another compute replica](#) is significantly behind applying new log records from the Log service.

The following wait types (in [sys.dm\\_os\\_wait\\_stats](#)) describe the reasons why log rate can be throttled on the primary compute replica:

WAIT TYPE	DESCRIPTION
RBIO_RG_STORAGE	Occurs when a Hyperscale database primary compute node log generation rate is being throttled due to delayed log consumption at the page server(s).
RBIO_RG_DESTAGE	Occurs when a Hyperscale database compute node log generation rate is being throttled due to delayed log consumption by the long-term log storage.
RBIO_RG_REPLICA	Occurs when a Hyperscale database compute node log generation rate is being throttled due to delayed log consumption by the readable secondary replica(s).
RBIO_RG_LOCALDESTAGE	Occurs when a Hyperscale database compute node log generation rate is being throttled due to delayed log consumption by the log service.

## Page server reads

The compute replicas do not cache a full copy of the database locally. The data local to the compute replica is stored in the Buffer Pool (in memory) and in the local Resilient Buffer Pool Extension (RBPEX) cache that is a partial (non-covering) cache of data pages. This local RBPEX cache is sized proportionally to the compute size and is three times the memory of the compute tier. RBPEX is similar to the Buffer Pool in that it has the most frequently accessed data. Each page server, on the other hand, has a covering RBPEX cache for the portion of the database it maintains.

When a read is issued on a compute replica, if the data doesn't exist in the Buffer Pool or local RBPEX cache, a `getPage(pageId, LSN)` function call is issued, and the page is fetched from the corresponding page server. Reads from page servers are remote reads and are thus slower than reads from the local RBPEX. When troubleshooting

IO-related performance problems, we need to be able to tell how many IOs were done via relatively slower remote page server reads.

Several DMVs and extended events have columns and fields that specify the number of remote reads from a page server, which can be compared against the total reads. Query store also captures remote reads as part of the query run time stats.

- Columns to report page server reads are available in execution DMVs and catalog views, such as:
  - [sys.dm\\_exec\\_requests](#)
  - [sys.dm\\_exec\\_query\\_stats](#)
  - [sys.dm\\_exec\\_procedure\\_stats](#)
  - [sys.dm\\_exec\\_trigger\\_stats](#)
  - [sys.query\\_store\\_runtime\\_stats](#)
- Page server reads are added to the following extended events:
  - `sql_statement_completed`
  - `sp_statement_completed`
  - `sql_batch_completed`
  - `rpc_completed`
  - `scan_stopped`
  - `query_store_begin_persist_runtime_stat`
  - `query-store_execution_runtime_info`
- ActualPageServerReads/ActualPageServerReadAheads are added to query plan XML for actual plans. For example:

```
<RunTimeCountersPerThread Thread="8" ActualRows="90466461" ActualRowsRead="90466461" Batches="0"
ActualEndOfScans="1" ActualExecutions="1" ActualExecutionMode="Row" ActualElapsedms="133645"
ActualCPUms="85105" ActualScans="1" ActualLogicalReads="6032256" ActualPhysicalReads="0"
ActualPageServerReads="0" ActualReadAheads="6027814" ActualPageServerReadAheads="5687297"
ActualLobLogicalReads="0" ActualLobPhysicalReads="0" ActualLobPageServerReads="0" ActualLobReadAheads="0"
ActualLobPageServerReadAheads="0" />
```

#### NOTE

To view these attributes in the query plan properties window, SSMS 18.3 or later is required.

## Virtual file stats and IO accounting

In Azure SQL Database, the [sys.dm\\_io\\_virtual\\_file\\_stats\(\)](#) DMF is the primary way to monitor SQL Server IO. IO characteristics in Hyperscale are different due to its [distributed architecture](#). In this section, we focus on IO (reads and writes) to data files as seen in this DMF. In Hyperscale, each data file visible in this DMF corresponds to a remote page server. The RBPEX cache mentioned here is a local SSD-based cache, that is a non-covering cache on the compute replica.

### Local RBPEX cache usage

Local RBPEX cache exists on the compute replica, on local SSD storage. Thus, IO against this cache is faster than IO against remote page servers. Currently, [sys.dm\\_io\\_virtual\\_file\\_stats\(\)](#) in a Hyperscale database has a special row reporting the IO against the local RBPEX cache on the compute replica. This row has the value of 0 for both `database_id` and `file_id` columns. For example, the query below returns RBPEX usage statistics since database startup.

```
select * from sys.dm_io_virtual_file_stats(0,NULL);
```

A ratio of reads done on RBPEX to aggregated reads done on all other data files provides RBPEX cache hit ratio.

### Data reads

- When reads are issued by the SQL Server engine on a compute replica, they may be served either by the local RBPEX cache, or by remote page servers, or by a combination of the two if reading multiple pages.
- When the compute replica reads some pages from a specific file, for example file\_id 1, if this data resides solely on the local RBPEX cache, all IO for this read is accounted against file\_id 0 (RBPEX). If some part of that data is in the local RBPEX cache, and some part is on a remote page server, then IO is accounted towards file\_id 0 for the part served from RBPEX, and the part served from the remote page server is accounted towards file\_id 1.
- When a compute replica requests a page at a particular [LSN](#) from a page server, if the page server has not caught up to the LSN requested, the read on the compute replica will wait until the page server catches up before the page is returned to the compute replica. For any read from a page server on the compute replica, you will see the `PAGEIOLATCH_*` wait type if it is waiting on that IO. In Hyperscale, this wait time includes both the time to catch up the requested page on the page server to the LSN required, and the time needed to transfer the page from the page server to the compute replica.
- Large reads such as read-ahead are often done using "[Scatter-Gather](#)" Reads. This allows reads of up to 4 MB of pages at a time, considered a single read in the SQL Server engine. However, when data being read is in RBPEX, these reads are accounted as multiple individual 8 KB reads, since the buffer pool and RBPEX always use 8 KB pages. As the result, the number of read IOs seen against RBPEX may be larger than the actual number of IOs performed by the engine.

## Data writes

- The primary compute replica does not write directly to page servers. Instead, log records from the Log service are replayed on corresponding page servers.
- Writes that happen on the compute replica are predominantly writes to the local RBPEX (file\_id 0). For writes on logical files that are larger than 8 KB, in other words those done using [Gather-write](#), each write operation is translated into multiple 8 KB individual writes to RBPEX since the buffer pool and RBPEX always use 8 KB pages. As the result, the number of write IOs seen against RBPEX may be larger than the actual number of IOs performed by the engine.
- Non-RBPEX files, or data files other than file\_id 0 that correspond to page servers, also show writes. In the Hyperscale service tier, these writes are simulated, because the compute replicas never write directly to page servers. Write IOPS and throughput are accounted as they occur on the compute replica, but latency for data files other than file\_id 0 does not reflect the actual latency of page server writes.

## Log writes

- On the primary compute, a log write is accounted for in file\_id 2 of `sys.dm_io_virtual_file_stats`. A log write on primary compute is a write to the log Landing Zone.
- Log records are not hardened on the secondary replica on a commit. In Hyperscale, log is applied by the Log service to the secondary replicas asynchronously. Because log writes don't actually occur on secondary replicas, any accounting of Log IOs on the secondary replicas is for tracking purposes only.

## Data IO in resource utilization statistics

In a non-Hyperscale database, combined read and write IOPS against data files, relative to the [resource governance](#) data IOPS limit, are reported in `sys.dm_db_resource_stats` and `sys.resource_stats` views, in the `avg_data_io_percent` column. The same value is reported in the portal as *Data IO Percentage*.

In a Hyperscale database, this column reports on data IOPS utilization relative to the limit for local storage on compute replica only, specifically IO against RBPEX and `tempdb`. A 100% value in this column indicates that resource governance is limiting local storage IOPS. If this is correlated with a performance problem, tune the workload to generate less IO, or increase database service objective to increase the resource governance *Max Data IOPS limit*. For resource governance of RBPEX reads and writes, the system counts individual 8 KB IOs, rather than larger IOs that may be issued by the SQL Server engine.

Data IO against remote page servers is not reported in resource utilization views or in the portal, but is reported in the `sys.dm_io_virtual_file_stats()` DMF, as noted earlier.

## Additional resources

- For vCore resource limits for a Hyperscale single database see [Hyperscale service tier vCore Limits](#)
- For Azure SQL Database performance tuning, see [Query performance in Azure SQL Database](#)
- For performance tuning using Query Store, see [Performance monitoring using Query store](#)
- For DMV monitoring scripts, see [Monitoring performance Azure SQL Database using dynamic management views](#)

# Resource management in dense elastic pools

11/19/2019 • 11 minutes to read • [Edit Online](#)

Azure SQL Database [elastic pools](#) is a cost-effective solution for managing many databases with varying resource usage. All databases in an elastic pool share the same allocation of resources, such as CPU, memory, worker threads, storage space, tempdb, on the assumption that only a subset of databases in the pool will use compute resources at any given time. This assumption allows elastic pools to be cost-effective. Instead of paying for all resources each individual database could potentially need, customers pay for a much smaller set of resources, shared among all databases in the pool.

## Resource governance

Resource sharing requires the system to carefully control resource usage to minimize the "noisy neighbor" effect, where a database with high resource consumption affects other databases in the same elastic pool. At the same time, the system must provide sufficient resources for features such as high availability and disaster recovery (HADR), backup and restore, monitoring, Query Store, and Automatic tuning to function reliably.

Azure SQL Database achieves these goals by using multiple resource governance mechanisms, including Windows [Job Objects](#) for process level resource governance, Windows [File Server Resource Manager \(FSRM\)](#) for storage quota management, and a modified and extended version of SQL Server [Resource Governor](#) to implement resource governance within each SQL Server instance running in Azure SQL Database.

The overarching design goal of elastic pools is to be cost-effective. For this reason, the system intentionally allows customers to create *dense* pools, that is pools with the number of databases approaching or at the maximum allowed, but with a moderate allocation of compute resources. For the same reason, the system doesn't reserve all potentially needed resources for its internal processes, but allows resource sharing between internal processes and user workloads.

This approach allows customers to use dense elastic pools to achieve adequate performance and major cost savings. However, if the workload against databases in a dense pool is sufficiently intense, resource contention becomes significant. Resource contention reduces user workload performance, and can negatively impact internal processes.

When resource contention occurs in a densely packed pool, customers can choose one or more of the following actions to mitigate it:

- Tune query workload to reduce resource consumption.
- Reduce pool density by moving some databases to another pool, or by making them standalone databases.
- Scale up the pool to get more resources.

For suggestions on how to implement the last two actions, see [Operational recommendations](#) later in this article. Reducing resource contention benefits both user workloads and internal processes, and lets the system reliably maintain expected level of service.

## Monitoring resource consumption

To avoid performance degradation due to resource contention, customers using dense elastic pools should proactively monitor resource consumption, and take timely action if increasing resource contention starts affecting workloads. Continuous monitoring is important because resource usage in a pool changes over time, due to changes in user workload, changes in data volumes and distribution, changes in pool density, and changes in the SQL Server database engine.

Azure SQL Database provides several metrics that are relevant for this type of monitoring. Exceeding the recommended average value for each metric indicates resource contention in the pool, and should be addressed using one of the actions mentioned earlier.

METRIC NAME	DESCRIPTION	RECOMMENDED AVERAGE VALUE
<code>avg_instance_cpu_percent</code>	CPU utilization of the SQL Server process associated with an elastic pool, as measured by the underlying operating system. Available in the <code>sys.dm_db_resource_stats</code> view in every database, and in the <code>sys.elastic_pool_resource_stats</code> view in the <code>master</code> database. This metric is also emitted to Azure Monitor, where it is named <code>sqlserver_process_core_percent</code> , and can be viewed in Azure portal. This value is the same for every database in the same elastic pool.	Below 70%. Occasional short spikes up to 90% may be acceptable.
<code>max_worker_percent</code>	<b>Worker thread</b> utilization. Provided for each database in the pool, as well as for the pool itself. There are different limits on the number of worker threads at the database level, and at the pool level, therefore monitoring this metric at both levels is recommended. Available in the <code>sys.dm_db_resource_stats</code> view in every database, and in the <code>sys.elastic_pool_resource_stats</code> view in the <code>master</code> database. This metric is also emitted to Azure Monitor, where it is named <code>workers_percent</code> , and can be viewed in Azure portal.	Below 80%. Spikes up to 100% will cause connection attempts and queries to fail.
<code>avg_data_io_percent</code>	IOPS utilization for read and write physical IO. Provided for each database in the pool, as well as for the pool itself. There are different limits on the number of IOPS at the database level, and at the pool level, therefore monitoring this metric at both levels is recommended. Available in the <code>sys.dm_db_resource_stats</code> view in every database, and in the <code>sys.elastic_pool_resource_stats</code> view in the <code>master</code> database. This metric is also emitted to Azure Monitor, where it is named <code>physical_data_read_percent</code> , and can be viewed in Azure portal.	Below 80%. Occasional short spikes up to 100% may be acceptable.

METRIC NAME	DESCRIPTION	RECOMMENDED AVERAGE VALUE
<code>avg_log_write_percent</code>	<p>Throughput utilizations for transaction log write IO. Provided for each database in the pool, as well as for the pool itself. There are different limits on the log throughput at the database level, and at the pool level, therefore monitoring this metric at both levels is recommended. Available in the <a href="#">sys.dm_db_resource_stats</a> view in every database, and in the <a href="#">sys.elastic_pool_resource_stats</a> view in the <code>master</code> database. This metric is also emitted to Azure Monitor, where it is named <code>log_write_percent</code>, and can be viewed in Azure portal. When this metric is close to 100%, all database modifications (INSERT, UPDATE, DELETE, MERGE statements, SELECT ... INTO, BULK INSERT, etc.) will be slower.</p>	Below 90%. Occasional short spikes up to 100% may be acceptable.
<code>oom_per_second</code>	<p>The rate of out-of-memory (OOM) errors in an elastic pool, which is an indicator of memory pressure. Available in the <a href="#">sys.dm_resource_governor_resource_pools_history_ex</a> view. See <a href="#">Examples</a> for a sample query to calculate this metric.</p>	0
<code>avg_storage_percent</code>	<p>Storage space utilization at the elastic pool level. Available in the <a href="#">sys.elastic_pool_resource_stats</a> view in the <code>master</code> database. This metric is also emitted to Azure Monitor, where it is named <code>storage_percent</code>, and can be viewed in Azure portal.</p>	Below 80%. Can approach 100% for pools with no data growth.
<code>tempdb_log_used_percent</code>	<p>Transaction log space utilization in the <code>tempdb</code> database. Even though temporary objects created in one database are not visible in other databases in the same elastic pool, <code>tempdb</code> is a shared resource for all databases in the same pool. A long running or idle transaction in <code>tempdb</code> started from one database in the pool can consume a large portion of transaction log, and cause failures for queries in other databases in the same pool. Available in the <a href="#">sys.dm_db_log_space_usage</a> view. This metric is also emitted to Azure Monitor, and can be viewed in Azure portal. See <a href="#">Examples</a> for a sample query to return the current value of this metric.</p>	Below 50%. Occasional spikes up to 80% are acceptable.

In addition to these metrics, Azure SQL Database provides a view that returns actual resource governance limits, as well as views that return resource utilization statistics at the resource pool level, and at the workload group level.

VIEW NAME	DESCRIPTION
<a href="#">sys.dm_user_db_resource_governance</a>	Returns actual configuration and capacity settings used by resource governance mechanisms in the current database or elastic pool.
<a href="#">sys.dm_resource_governor_resource_pools</a>	Returns information about the current resource pool state, the current configuration of resource pools, and cumulative resource pool statistics.
<a href="#">sys.dm_resource_governor_workload_groups</a>	Returns cumulative workload group statistics and the current configuration of the workload group. This view can be joined with <a href="#">sys.dm_resource_governor_resource_pools</a> on the <code>pool_id</code> column to get resource pool information.
<a href="#">sys.dm_resource_governor_resource_pools_history_ex</a>	Returns resource pool utilization statistics for the last 32 minutes. Each row represents a 20-second interval. The <code>delta_</code> columns return the change in each statistic during the interval.
<a href="#">sys.dm_resource_governor_workload_groups_history_ex</a>	Returns workload group utilization statistics for the last 32 minutes. Each row represents a 20-second interval. The <code>delta_</code> columns return the change in each statistic during the interval.

These views can be used to monitor resource utilization and troubleshoot resource contention in near real-time. User workload on the primary and readable secondary replicas, including geo-replicas, is classified into the `sloSharedPool1` resource pool and `UserPrimaryGroup.DBId[N]` workload group, where `N` stands for the database ID value.

In addition to monitoring current resource utilization, customers using dense pools can maintain historical resource utilization data in a separate data store. This data can be used in predictive analysis to proactively manage resource utilization based on historical and seasonal trends.

## Operational recommendations

**Leave sufficient resource headroom.** If resource contention and performance degradation occurs, mitigation may involve moving some databases out of the affected elastic pool, or scaling up the pool, as noted earlier. However, these actions require additional compute resources to complete. In particular, for Premium and Business Critical pools, these actions require transferring all data for the databases being moved, or for all databases in the elastic pool if the pool is scaled up. Data transfer is a long running and resource-intensive operation. If the pool is already under high resource pressure, the mitigating operation itself will degrade performance even further. In extreme cases, it may not be possible to solve resource contention via database move or pool scale-up because the required resources are not available. In this case, temporarily reducing query workload on the affected elastic pool may be the only solution.

Customers using dense pools should closely monitor resource utilization trends as described earlier, and take mitigating action while metrics remain within the recommended ranges and there are still sufficient resources in the elastic pool.

Resource utilization depends on multiple factors that change over time for each database and each elastic pool. Achieving optimal price/performance ratio in dense pools requires continuous monitoring and rebalancing, that is moving databases from more utilized pools to less utilized pools, and creating new pools as necessary to accommodate increased workload.

**Do not move "hot" databases.** If resource contention at the pool level is primarily caused by a small number of highly utilized databases, it may be tempting to move these databases to a less utilized pool, or make them standalone databases. However, doing this while a database remains highly utilized is not recommended, because the move operation will further degrade performance, both for the database being moved, and for the entire pool. Instead, either wait until high utilization subsides, or move less utilized databases instead to relieve resource pressure at the pool level. But moving databases with very low utilization does not provide any benefit in this case, because it does not materially reduce resource utilization at the pool level.

**Create new databases in a "quarantine" pool.** In scenarios where new databases are created frequently, such as applications using the tenant-per-database model, there is risk that a new database placed into an existing elastic pool will unexpectedly consume significant resources and affect other databases and internal processes in the pool. To mitigate this risk, create a separate "quarantine" pool with ample allocation of resources. Use this pool for new databases with yet unknown resource consumption patterns. Once a database has stayed in this pool for a business cycle, such as a week or a month, and its resource consumption is known, it can be moved to a pool with sufficient capacity to accommodate this additional resource usage.

**Avoid overly dense logical servers.** Azure SQL Database [supports](#) up to 5000 databases per logical server. Customers using elastic pools with thousands of databases may consider placing multiple elastic pools on a single server, with the total number of databases up to the supported limit. However, logical servers with many thousands of databases create operational challenges. Operations that require enumerating all databases on a server, for example viewing databases in the portal, will be slower. Operational errors, such as incorrect modification of server level logins or firewall rules, will affect a larger number of databases. Accidental deletion of the logical server will require assistance from Microsoft Support to recover databases on the deleted server, and will cause a prolonged outage for all affected databases.

We recommend limiting the number of databases per logical server to a lower number than the maximum supported. In many scenarios, using up to 1000-2000 databases per server is optimal. To reduce the likelihood of accidental server deletion, we recommend placing a [delete lock](#) on the logical server or its resource group.

In the past, certain scenarios involving moving databases in, out, or between elastic pools on the same logical server were faster than when moving databases between logical servers. Currently, all database moves execute at the same speed regardless of source and destination logical server.

## Examples

### Monitoring memory utilization

This query calculates the `oom_per_second` metric for each resource pool, over the last 32 minutes. This query can be executed in any database in an elastic pool.

```
SELECT pool_id,
 name AS resource_pool_name,
 IIF(name LIKE 'SloSharedPool%' OR name LIKE 'UserPool%', 'user', 'system') AS resource_pool_type,
 SUM(CAST(delta_out_of_memory_count AS decimal))/(SUM(duration_ms)/1000.) AS oom_per_second
 FROM sys.dm_resource_governor_resource_pools_history_ex
 GROUP BY pool_id, name
 ORDER BY pool_id;
```

### Monitoring `tempdb` log space utilization

This query returns the current value of the `tempdb_log_used_percent` metric. This query can be executed in any database in an elastic pool.

```
SELECT used_log_space_in_percent AS tempdb_log_used_percent
 FROM tempdb.sys.dm_db_log_space_usage;
```

## Next steps

- For an introduction to elastic pools, see [Elastic pools help you manage and scale multiple Azure SQL databases](#).
- For information on tuning query workloads to reduce resource utilization, see [Monitoring and tuning](#), and [Monitoring and performance tuning](#).

# Dynamically scale database resources with minimal downtime

1/28/2020 • 4 minutes to read • [Edit Online](#)

Azure SQL Database enables you to dynamically add more resources to your database with minimal [downtime](#); however, there is a switch over period where connectivity is lost to the database for a short amount of time, which can be mitigated using retry logic.

## Overview

When demand for your app grows from a handful of devices and customers to millions, Azure SQL Database scales on the fly with minimal downtime. Scalability is one of the most important characteristics of PaaS that enables you to dynamically add more resources to your service when needed. Azure SQL Database enables you to easily change resources (CPU power, memory, IO throughput, and storage) allocated to your databases.

You can mitigate performance issues due to increased usage of your application that cannot be fixed using indexing or query rewrite methods. Adding more resources enables you to quickly react when your database hits the current resource limits and needs more power to handle the incoming workload. Azure SQL Database also enables you to scale-down the resources when they are not needed to lower the cost.

You don't need to worry about purchasing hardware and changing underlying infrastructure. Scaling database can be easily done via Azure portal using a slider.

Azure SQL Database offers the [DTU-based purchasing model](#) and the [vCore-based purchasing model](#).

- The [DTU-based purchasing model](#) offers a blend of compute, memory, and IO resources in three service tiers to support lightweight to heavyweight database workloads: Basic, Standard, and Premium. Performance levels within each tier provide a different mix of these resources, to which you can add additional storage resources.
- The [vCore-based purchasing model](#) lets you choose the number of vCores, the amount or memory, and the amount and speed of storage. This purchasing model offers three service tiers: General Purpose, Business Critical, and Hyperscale.

You can build your first app on a small, single database at a low cost per month in the Basic, Standard, or General Purpose service tier and then change its service tier manually or programmatically at any time to the Premium or Business Critical service tier to meet the needs of your solution. You can adjust performance without downtime to your app or to your customers. Dynamic scalability enables your database to transparently respond to rapidly changing resource requirements and enables you to only pay for the resources that you need when you need them.

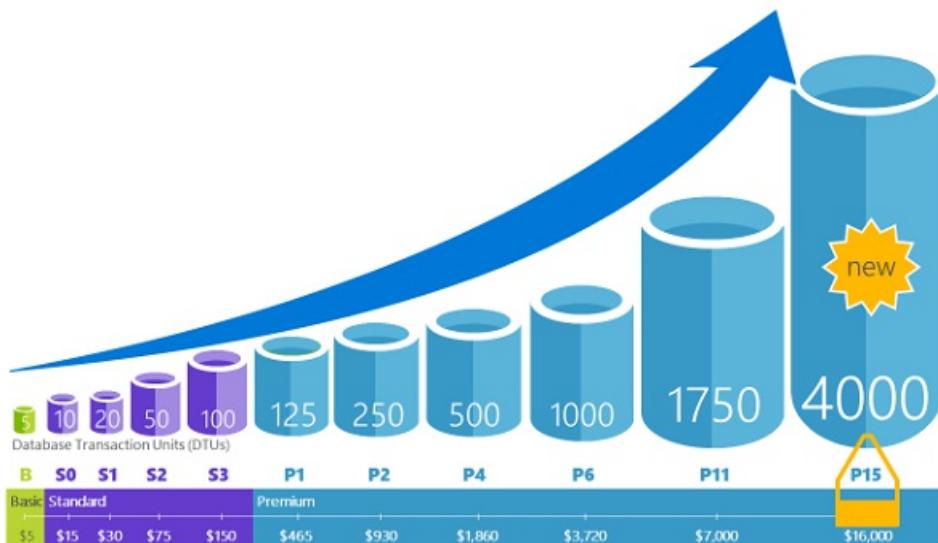
### NOTE

Dynamic scalability is different from autoscale. Autoscale is when a service scales automatically based on criteria, whereas dynamic scalability allows for manual scaling with a minimal downtime.

Single Azure SQL Database supports manual dynamic scalability, but not autoscale. For a more *automatic* experience, consider using elastic pools, which allow databases to share resources in a pool based on individual database needs. However, there are scripts that can help automate scalability for a single Azure SQL Database. For an example, see [Use PowerShell to monitor and scale a single SQL Database](#).

You can change [DTU service tiers](#) or [vCore characteristics](#) at any time with minimal downtime to your application

(generally averaging under four seconds). For many businesses and apps, being able to create databases and dial performance up or down on demand is enough, especially if usage patterns are relatively predictable. But if you have unpredictable usage patterns, it can make it hard to manage costs and your business model. For this scenario, you use an elastic pool with a certain number of eDTUs that are shared among multiple databases in the pool.



All three flavors of Azure SQL Database offer some ability to dynamically scale your databases:

- With a [single database](#), you can use either [DTU](#) or [vCore](#) models to define maximum amount of resources that will be assigned to each database.
- A [Managed Instance](#) uses [vCores](#) mode and enables you to define maximum CPU cores and maximum of storage allocated to your instance. All databases within the instance will share the resources allocated to the instance.
- [Elastic pools](#) enable you to define maximum resource limit per group of databases in the pool.

Initiating scale up or scale down action in any of the flavors would restart database engine process and move it to a different virtual machine if needed. Moving database engine process to a new virtual machine is [online process](#) where you can continue using your existing Azure SQL Database service while the process is in progress. Once the target database engine is fully initialized and ready to process the queries, the connections will be [switched from source to target database engine](#).

#### NOTE

You can expect a short connection break when the scale up/scale down process is finished. If you have implemented [Retry logic for standard transient errors](#), you will not notice the failover.

## Alternative scale methods

Scaling resources is the easiest and the most effective way to improve performance of your database without changing either the database or application code. In some cases, even the highest service tiers, compute sizes, and performance optimizations might not handle your workload in a successful and cost-effective way. In that case you have these additional options to scale your database:

- [Read scale-out](#) is a available feature where you are getting one read-only replica of your data where you can execute demanding read-only queries such as reports. Read-only replica will handle your read-only workload without affecting resource usage on your primary database.
- [Database sharding](#) is a set of techniques that enables you to split your data into several databases and scale them independently.

## Next steps

- For information about improving database performance by changing database code, see [Find and apply performance recommendations](#).
- For information about letting built-in database intelligence optimize your database, see [Automatic tuning](#).
- For information about Read Scale-out in the Azure SQL Database service, see how to [use read-only replicas to load balance read-only query workloads](#).
- For information about a Database sharding, see [Scaling out with Azure SQL Database](#).

# Use read-only replicas to load-balance read-only query workloads

11/22/2019 • 7 minutes to read • [Edit Online](#)

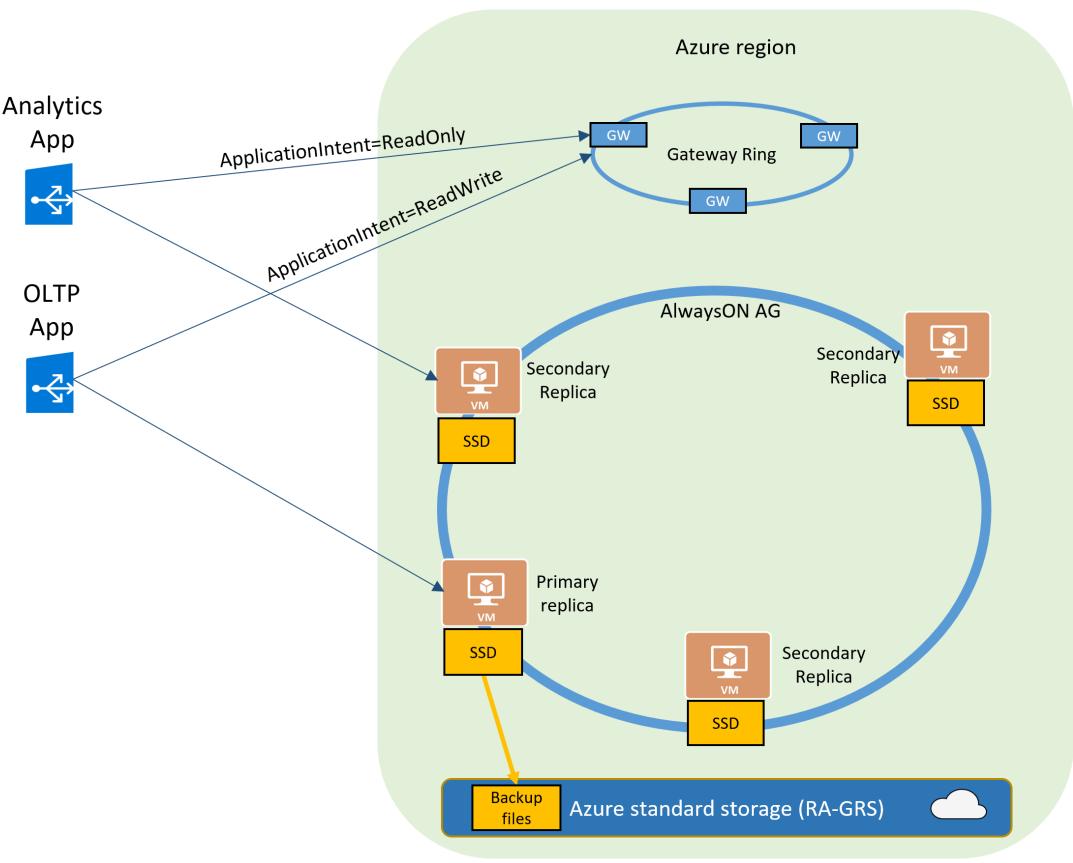
## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

As part of the [High Availability architecture](#), each database in the Premium and Business Critical service tier is automatically provisioned with a primary replica and several secondary replicas. The secondary replicas are provisioned with the same compute size as the primary replica. The **Read Scale-Out** feature allows you to load-balance SQL Database read-only workloads using the capacity of one of the read-only replicas instead of sharing the read-write replica. This way the read-only workload will be isolated from the main read-write workload and will not affect its performance. The feature is intended for the applications that include logically separated read-only workloads, such as analytics. In the Premium and Business Critical service tiers, applications could gain performance benefits using this additional capacity at no extra cost.

The **Read Scale-Out** feature is also available in the Hyperscale service tier when at least one secondary replica is created. Multiple secondary replicas can be used if read-only workloads require more resources than available on one secondary replica. The High Availability architecture of Basic, Standard, and General Purpose service tiers does not include any replicas. The **Read Scale-Out** feature is not available in these service tiers.

The following diagram illustrates it using a Business Critical database.



The Read Scale-Out feature is enabled by default on new Premium, Business Critical, and Hyperscale databases. For Hyperscale, one secondary replica is created by default for new databases. If your SQL connection string is configured with `ApplicationIntent=ReadOnly`, the application will be redirected by the gateway to a read-only replica of that database. For information on how to use the `ApplicationIntent` property, see [Specifying Application Intent](#).

If you wish to ensure that the application connects to the primary replica regardless of the `ApplicationIntent` setting in the SQL connection string, you must explicitly disable read scale-out when creating the database or when altering its configuration. For example, if you upgrade your database from Standard or General Purpose tier to Premium, Business Critical or Hyperscale tier and want to make sure all your connections continue to go to the primary replica, disable Read Scale-out. For details on how to disable it, see [Enable and disable Read Scale-Out](#).

#### **NOTE**

Query Data Store, Extended Events, SQL Profiler and Audit features are not supported on the read-only replicas.

## Data consistency

One of the benefits of replicas is that the replicas are always in the transactionally consistent state, but at different points in time there may be some small latency between different replicas. Read Scale-Out supports session-level consistency. It means, if the read-only session reconnects after a connection error caused by replica unavailability, it may be redirected to a replica that is not 100% up-to-date with the read-write replica. Likewise, if an application writes data using a read-write session and immediately reads it using a read-only session, it is possible that the latest updates are not immediately visible on the replica. The latency is caused by an asynchronous transaction log redo operation.

#### NOTE

Replication latencies within the region are low and this situation is rare.

## Connect to a read-only replica

When you enable Read Scale-Out for a database, the `ApplicationIntent` option in the connection string provided by the client dictates whether the connection is routed to the write replica or to a read-only replica. Specifically, if the `ApplicationIntent` value is `ReadWrite` (the default value), the connection will be directed to the database's read-write replica. This is identical to existing behavior. If the `ApplicationIntent` value is `ReadOnly`, the connection is routed to a read-only replica.

For example, the following connection string connects the client to a read-only replica (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
Server=tcp:<server>.database.windows.net;Database=<mydatabase>;ApplicationIntent=ReadOnly;User ID=<myLogin>;Password=<myPassword>;Trusted_Connection=False; Encrypt=True;
```

Either of the following connection strings connects the client to a read-write replica (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
Server=tcp:<server>.database.windows.net;Database=<mydatabase>;ApplicationIntent=ReadWrite;User ID=<myLogin>;Password=<myPassword>;Trusted_Connection=False; Encrypt=True;

Server=tcp:<server>.database.windows.net;Database=<mydatabase>;User ID=<myLogin>;Password=<myPassword>;Trusted_Connection=False; Encrypt=True;
```

## Verify that a connection is to a read-only replica

You can verify whether you are connected to a read-only replica by running the following query. It will return `READ_ONLY` when connected to a read-only replica.

```
SELECT DATABASEPROPERTYEX(DB_NAME(), 'Updateability')
```

#### NOTE

At any given time only one of the AlwaysON replicas is accessible by the `ReadOnly` sessions.

## Monitoring and troubleshooting read-only replica

When connected to a read-only replica, you can access the performance metrics using the `sys.dm_db_resource_stats` DMV. To access query plan statistics, use the `sys.dm_exec_query_stats`, `sys.dm_exec_query_plan` and `sys.dm_exec_sql_text` DMVs.

#### NOTE

The DMV `sys.resource_stats` in the logical master database returns CPU usage and storage data of the primary replica.

# Enable and disable Read Scale-Out

Read Scale-Out is enabled by default on Premium, Business Critical and Hyperscale service tiers. Read Scale-Out cannot be enabled in Basic, Standard, or General Purpose service tiers. Read Scale-Out is automatically disabled on Hyperscale databases configured with 0 replicas.

You can disable and re-enable Read Scale-Out on single databases and elastic pool databases in Premium or Business Critical service tier using the following methods.

## NOTE

The ability to disable Read Scale-Out is provided for backward compatibility.

## Azure portal

You can manage the Read Scale-out setting on the **Configure** database blade.

## PowerShell

### IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

Managing Read Scale-Out in Azure PowerShell requires the December 2016 Azure PowerShell release or newer. For the newest PowerShell release, see [Azure PowerShell](#).

You can disable or re-enable Read Scale-Out in Azure PowerShell by invoking the `Set-AzSqlDatabase` cmdlet and passing in the desired value – `Enabled` or `Disabled` -- for the `-ReadScale` parameter.

To disable read scale-out on an existing database (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
Set-AzSqlDatabase -ResourceGroupName <resourceGroupName> -ServerName <serverName> -DatabaseName <databaseName> -ReadScale Disabled
```

To disable read scale-out on a new database (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
New-AzSqlDatabase -ResourceGroupName <resourceGroupName> -ServerName <serverName> -DatabaseName <databaseName> -ReadScale Disabled -Edition Premium
```

To re-enable read scale-out on an existing database (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
Set-AzSqlDatabase -ResourceGroupName <resourceGroupName> -ServerName <serverName> -DatabaseName <databaseName> -ReadScale Enabled
```

## REST API

To create a database with read scale-out disabled, or to change the setting for an existing database, use the following method with the `readScale` property set to `Enabled` or `Disabled` as in the below sample request.

```
Method: PUT
URL:
https://management.azure.com/subscriptions/{SubscriptionId}/resourceGroups/{GroupName}/providers/Microsoft.SQL/servers/{ServerName}/databases/{DatabaseName}?api-version= 2014-04-01-preview
Body: {
 "properties": {
 "readScale": "Disabled"
 }
}
```

For more information, see [Databases - Create or Update](#).

## Using TempDB on read-only replica

The TempDB database is not replicated to the read-only replicas. Each replica has its own version of TempDB database that is created when the replica is created. It ensures that TempDB is updateable and can be modified during your query execution. If your read-only workload depends on using TempDB objects, you should create these objects as part of your query script.

## Using Read Scale-Out with geo-replicated databases

If you are using Read Scale-Out to load-balance read-only workloads on a database that is geo-replicated (for example, as a member of a failover group), make sure that read scale-out is enabled on both the primary and the geo-replicated secondary databases. This configuration will ensure that the same load-balancing experience continues when your application connects to the new primary after failover. If you are connecting to the geo-replicated secondary database with read-scale enabled, your sessions with `ApplicationIntent=ReadOnly` will be routed to one of the replicas the same way we route connections on the primary database. The sessions without `ApplicationIntent=ReadOnly` will be routed to the primary replica of the geo-replicated secondary, which is also read-only. Because geo-replicated secondary database has a different endpoint than the primary database, historically to access the secondary it wasn't required to set `ApplicationIntent=ReadOnly`. To ensure backward compatibility, `sys.geo_replication_links` DMV shows `secondary_allow_connections=2` (any client connection is allowed).

### NOTE

Round-robin or any other load-balanced routing between the local replicas of the secondary database is not supported.

## Next steps

- For information about SQL Database Hyperscale offering, see [Hyperscale service tier](#).

# Scaling out with Azure SQL Database

11/7/2019 • 6 minutes to read • [Edit Online](#)

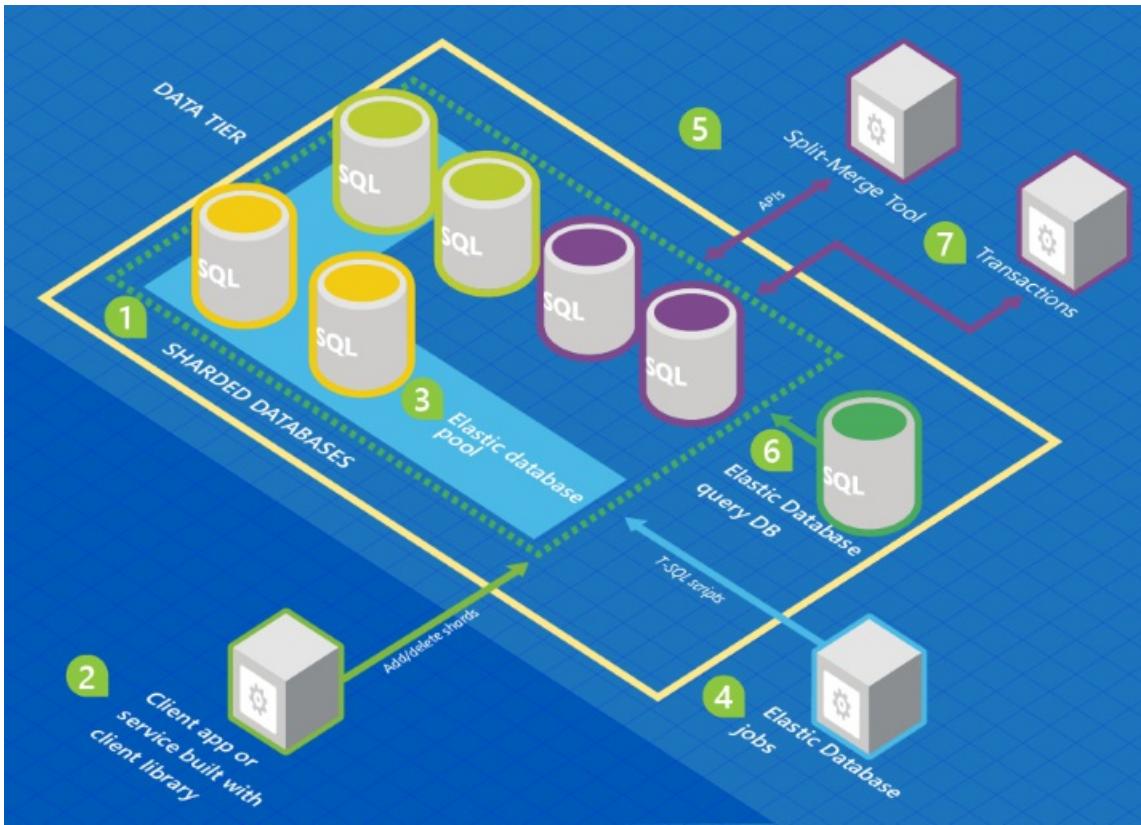
You can easily scale out Azure SQL databases using the **Elastic Database** tools. These tools and features let you use the database resources of **Azure SQL Database** to create solutions for transactional workloads, and especially Software as a Service (SaaS) applications. Elastic Database features are composed of the:

- [Elastic Database client library](#): The client library is a feature that allows you to create and maintain sharded databases. See [Get started with Elastic Database tools](#).
- [Elastic Database split-merge tool](#): moves data between sharded databases. This tool is useful for moving data from a multi-tenant database to a single-tenant database (or vice-versa). See [Elastic database Split-Merge tool tutorial](#).
- [Elastic Database jobs](#): Use jobs to manage large numbers of Azure SQL databases. Easily perform administrative operations such as schema changes, credentials management, reference data updates, performance data collection, or tenant (customer) telemetry collection using jobs.
- [Elastic Database query](#) (preview): Enables you to run a Transact-SQL query that spans multiple databases. This enables connection to reporting tools such as Excel, Power BI, Tableau, etc.
- [Elastic transactions](#): This feature allows you to run transactions that span several databases in Azure SQL Database. Elastic database transactions are available for .NET applications using ADO .NET and integrate with the familiar programming experience using the [System.Transaction classes](#).

The following graphic shows an architecture that includes the **Elastic Database features** in relation to a collection of databases.

In this graphic, colors of the database represent schemas. Databases with the same color share the same schema.

1. A set of **Azure SQL databases** is hosted on Azure using sharding architecture.
2. The **Elastic Database client library** is used to manage a shard set.
3. A subset of the databases is put into an **elastic pool**. (See [What is a pool?](#)).
4. An **Elastic Database job** runs scheduled or ad hoc T-SQL scripts against all databases.
5. The **split-merge tool** is used to move data from one shard to another.
6. The **Elastic Database query** allows you to write a query that spans all databases in the shard set.
7. **Elastic transactions** allow you to run transactions that span several databases.



## Why use the tools?

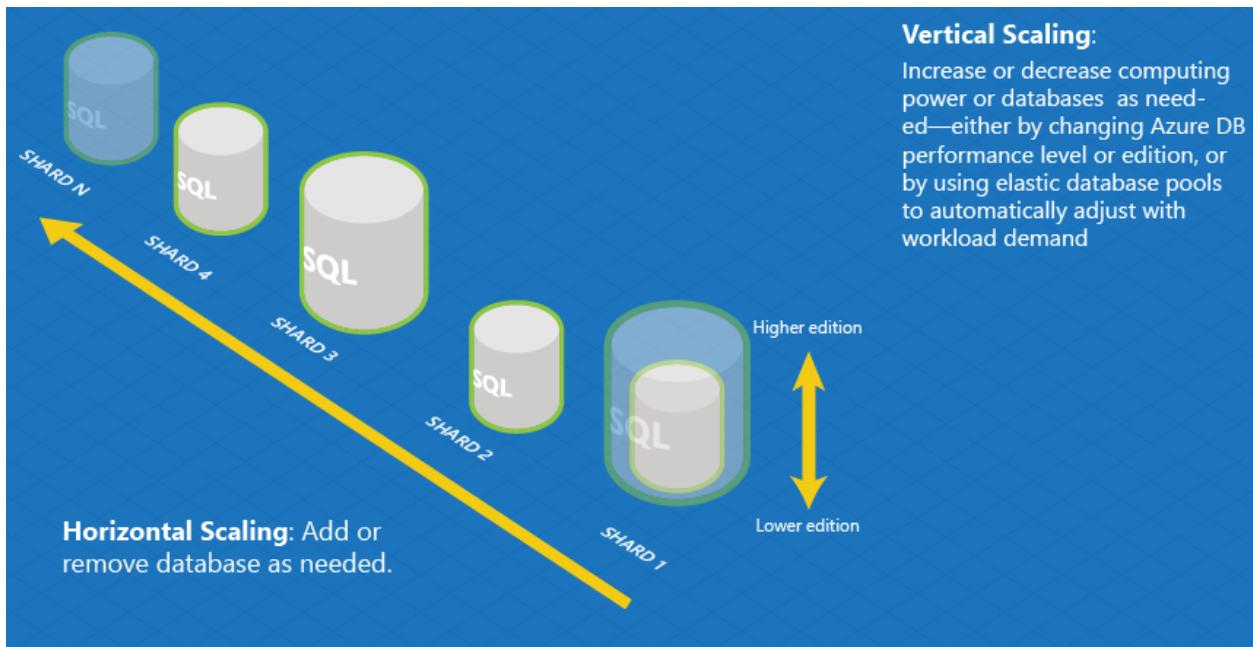
Achieving elasticity and scale for cloud applications has been straightforward for VMs and blob storage - simply add or subtract units, or increase power. But it has remained a challenge for stateful data processing in relational databases. Challenges emerged in these scenarios:

- Growing and shrinking capacity for the relational database part of your workload.
- Managing hotspots that may arise affecting a specific subset of data - such as a busy end-customer (tenant).

Traditionally, scenarios like these have been addressed by investing in larger-scale database servers to support the application. However, this option is limited in the cloud where all processing happens on predefined commodity hardware. Instead, distributing data and processing across many identically structured databases (a scale-out pattern known as "sharding") provides an alternative to traditional scale-up approaches both in terms of cost and elasticity.

## Horizontal and vertical scaling

The following figure shows the horizontal and vertical dimensions of scaling, which are the basic ways the elastic databases can be scaled.



Horizontal scaling refers to adding or removing databases in order to adjust capacity or overall performance, also called "scaling out". Sharding, in which data is partitioned across a collection of identically structured databases, is a common way to implement horizontal scaling.

Vertical scaling refers to increasing or decreasing the compute size of an individual database, also known as "scaling up."

Most cloud-scale database applications use a combination of these two strategies. For example, a Software as a Service application may use horizontal scaling to provision new end-customers and vertical scaling to allow each end-customer's database to grow or shrink resources as needed by the workload.

- Horizontal scaling is managed using the [Elastic Database client library](#).
- Vertical scaling is accomplished using Azure PowerShell cmdlets to change the service tier, or by placing databases in an elastic pool.

## Sharding

*Sharding* is a technique to distribute large amounts of identically structured data across a number of independent databases. It is especially popular with cloud developers creating Software as a Service (SAAS) offerings for end customers or businesses. These end customers are often referred to as "tenants". Sharding may be required for any number of reasons:

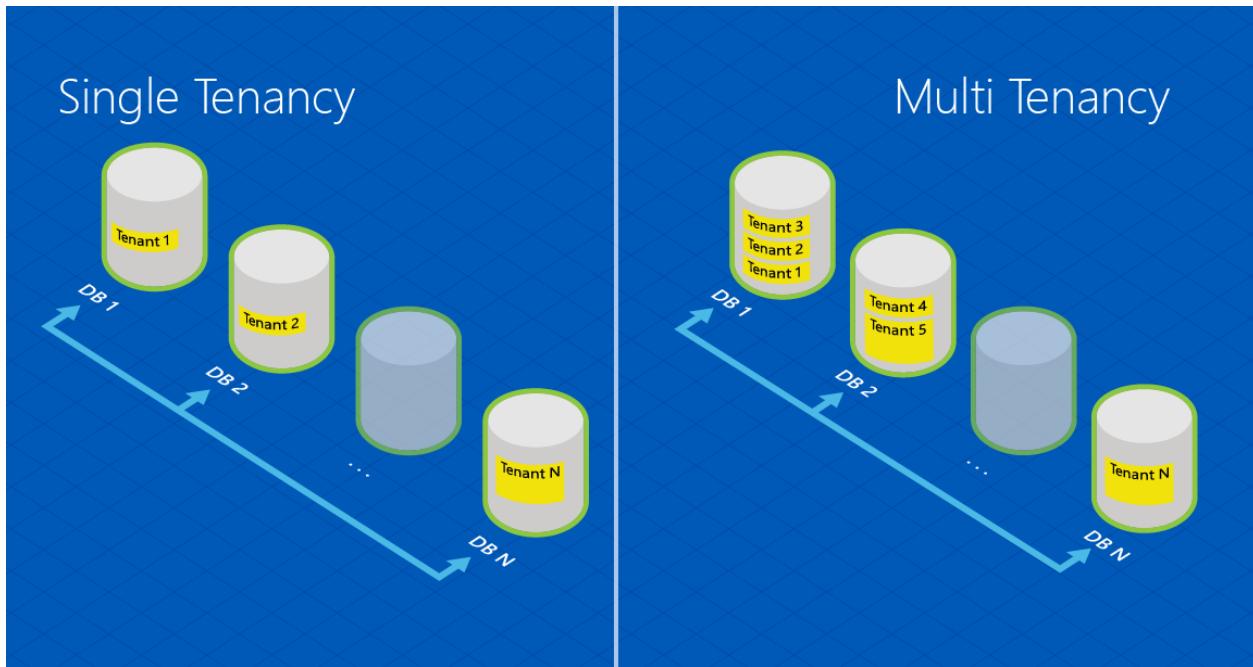
- The total amount of data is too large to fit within the constraints of an individual database
- The transaction throughput of the overall workload exceeds the capabilities of an individual database
- Tenants may require physical isolation from each other, so separate databases are needed for each tenant
- Different sections of a database may need to reside in different geographies for compliance, performance, or geopolitical reasons.

In other scenarios, such as ingestion of data from distributed devices, sharding can be used to fill a set of databases that are organized temporally. For example, a separate database can be dedicated to each day or week. In that case, the sharding key can be an integer representing the date (present in all rows of the sharded tables) and queries retrieving information for a date range must be routed by the application to the subset of databases covering the range in question.

Sharding works best when every transaction in an application can be restricted to a single value of a sharding key. That ensures that all transactions are local to a specific database.

## Multi-tenant and single-tenant

Some applications use the simplest approach of creating a separate database for each tenant. This approach is the **single tenant sharding pattern** that provides isolation, backup/restore ability, and resource scaling at the granularity of the tenant. With single tenant sharding, each database is associated with a specific tenant ID value (or customer key value), but that key need not always be present in the data itself. It is the application's responsibility to route each request to the appropriate database - and the client library can simplify this.



Others scenarios pack multiple tenants together into databases, rather than isolating them into separate databases. This pattern is a typical **multi-tenant sharding pattern** - and it may be driven by the fact that an application manages large numbers of small tenants. In multi-tenant sharding, the rows in the database tables are all designed to carry a key identifying the tenant ID or sharding key. Again, the application tier is responsible for routing a tenant's request to the appropriate database, and this can be supported by the elastic database client library. In addition, row-level security can be used to filter which rows each tenant can access - for details, see [Multi-tenant applications with elastic database tools and row-level security](#). Redistributing data among databases may be needed with the multi-tenant sharding pattern, and is facilitated by the elastic database split-merge tool. To learn more about design patterns for SaaS applications using elastic pools, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).

### Move data from multiple to single-tenancy databases

When creating a SaaS application, it is typical to offer prospective customers a trial version of the software. In this case, it is cost-effective to use a multi-tenant database for the data. However, when a prospect becomes a customer, a single-tenant database is better since it provides better performance. If the customer had created data during the trial period, use the [split-merge tool](#) to move the data from the multi-tenant to the new single-tenant database.

## Next steps

For a sample app that demonstrates the client library, see [Get started with Elastic Database tools](#).

To convert existing databases to use the tools, see [Migrate existing databases to scale out](#).

To see the specifics of the elastic pool, see [Price and performance considerations for an elastic pool](#), or create a new pool with [elastic pools](#).

## Additional resources

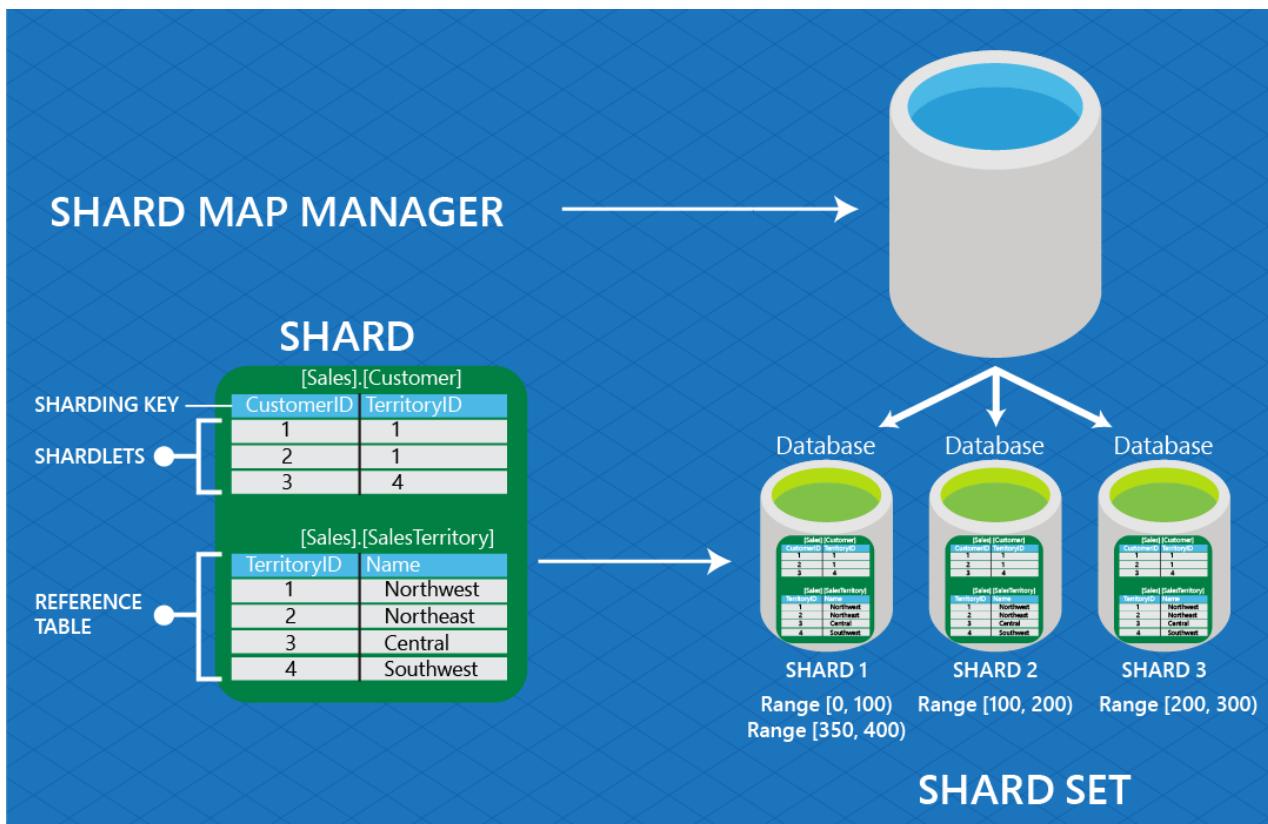
Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Elastic Database tools glossary

11/7/2019 • 2 minutes to read • [Edit Online](#)

The following terms are defined for the [Elastic Database tools](#), a feature of Azure SQL Database. The tools are used to manage [shard maps](#), and include the [client library](#), the [split-merge tool](#), [elastic pools](#), and [queries](#).

These terms are used in [Adding a shard using Elastic Database tools](#) and [Using the RecoveryManager class to fix shard map problems](#).



**Database:** An Azure SQL database.

**Data dependent routing:** The functionality that enables an application to connect to a shard given a specific sharding key. See [Data dependent routing](#). Compare to [Multi-Shard Query](#).

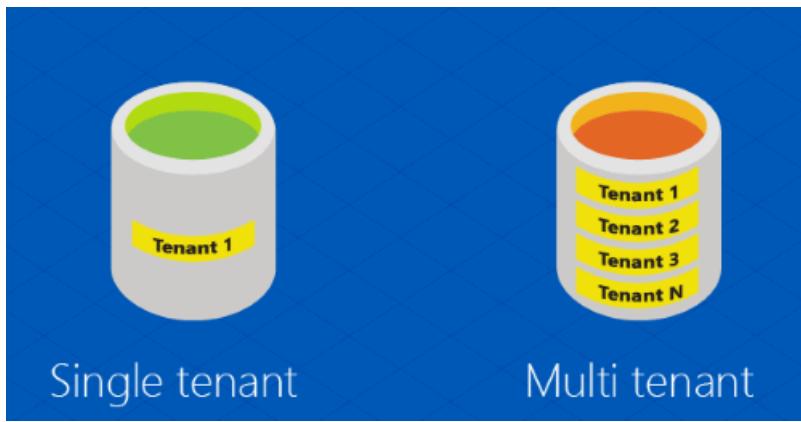
**Global shard map:** The map between sharding keys and their respective shards within a **shard set**. The global shard map is stored in the **shard map manager**. Compare to **local shard map**.

**List shard map:** A shard map in which sharding keys are mapped individually. Compare to [Range Shard Map](#).

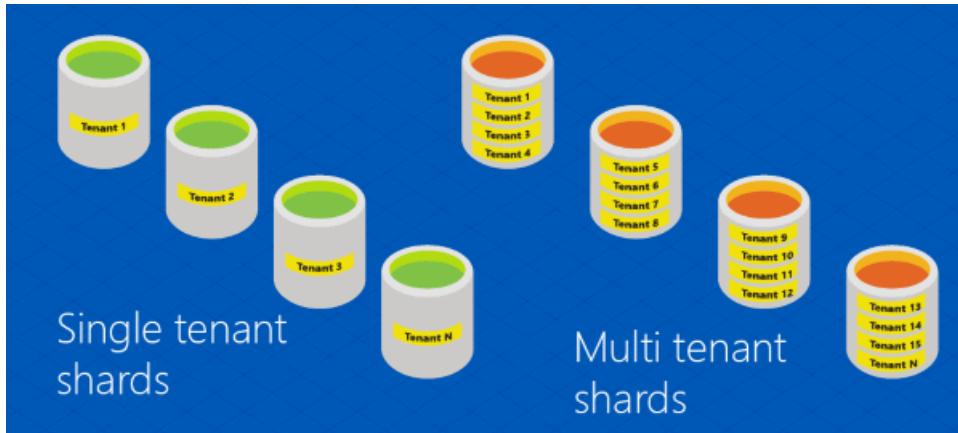
**Local shard map:** Stored on a shard, the local shard map contains mappings for the shardlets that reside on the shard.

**Multi-shard query:** The ability to issue a query against multiple shards; results sets are returned using UNION ALL semantics (also known as "fan-out query"). Compare to **data dependent routing**.

**Multi-tenant** and **Single-tenant**: This shows a single-tenant database and a multi-tenant database:



Here is a representation of **sharded** single and multi-tenant databases.



**Range shard map:** A shard map in which the shard distribution strategy is based on multiple ranges of contiguous values.

**Reference tables:** Tables that are not sharded but are replicated across shards. For example, zip codes can be stored in a reference table.

**Shard:** An Azure SQL database that stores data from a sharded data set.

**Shard elasticity:** The ability to perform both **horizontal scaling** and **vertical scaling**.

**Sharded tables:** Tables that are sharded, i.e., whose data is distributed across shards based on their sharding key values.

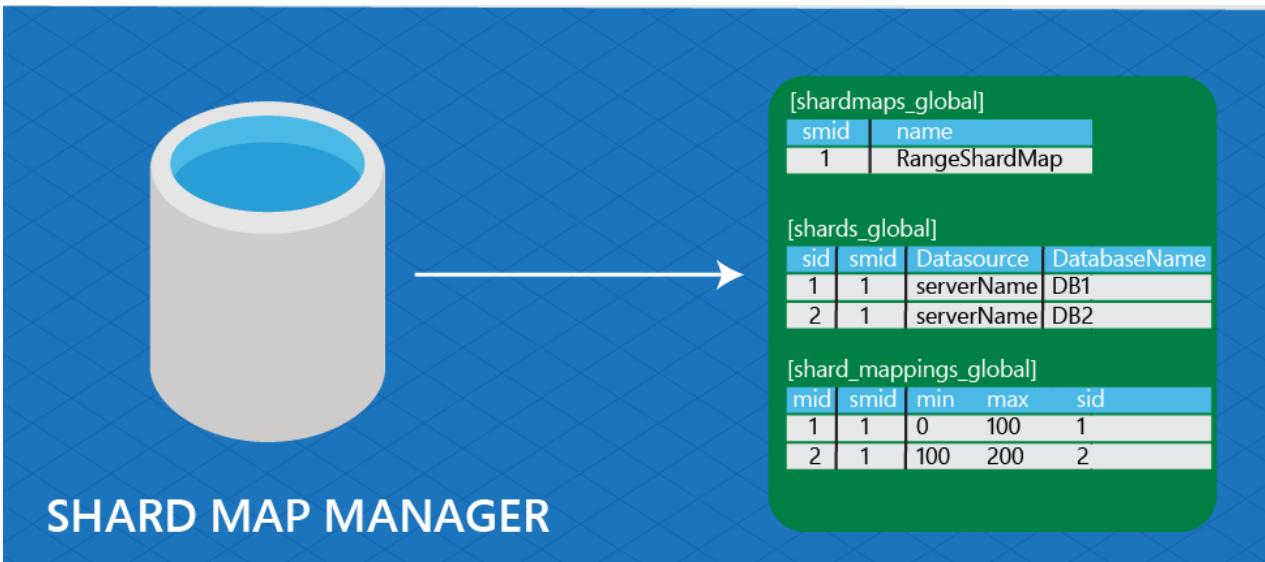
**Sharding key:** A column value that determines how data is distributed across shards. The value type can be one of the following: **int**, **bigint**, **varbinary**, or **uniqueidentifier**.

**Shard set:** The collection of shards that are attributed to the same shard map in the shard map manager.

**Shardlet:** All of the data associated with a single value of a sharding key on a shard. A shardlet is the smallest unit of data movement possible when redistributing sharded tables.

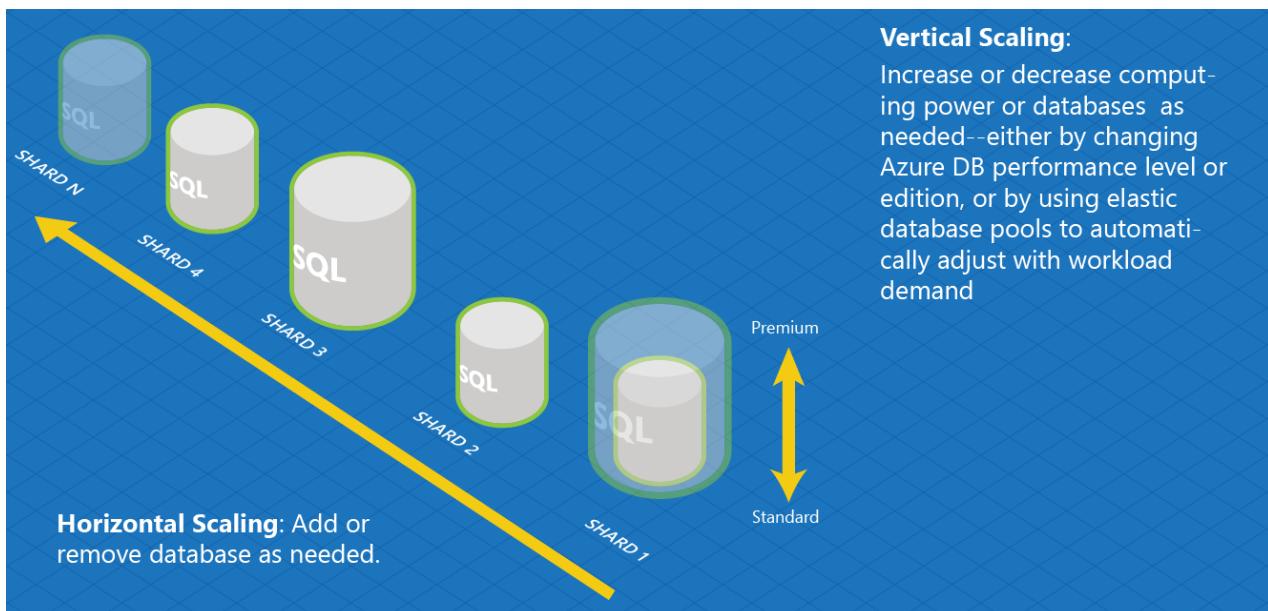
**Shard map:** The set of mappings between sharding keys and their respective shards.

**Shard map manager:** A management object and data store that contains the shard map(s), shard locations, and mappings for one or more shard sets.



## Verbs

**Horizontal scaling:** The act of scaling out (or in) a collection of shards by adding or removing shards to a shard map, as shown below.



**Merge:** The act of moving shardlets from two shards to one shard and updating the shard map accordingly.

**Shardlet move:** The act of moving a single shardlet to a different shard.

**Shard:** The act of horizontally partitioning identically structured data across multiple databases based on a sharding key.

**Split:** The act of moving several shardlets from one shard to another (typically new) shard. A sharding key is provided by the user as the split point.

**Vertical Scaling:** The act of scaling up (or down) the compute size of an individual shard. For example, changing a shard from Standard to Premium (which results in more computing resources).

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Building scalable cloud databases

11/7/2019 • 3 minutes to read • [Edit Online](#)

Scaling out databases can be easily accomplished using scalable tools and features for Azure SQL Database. In particular, you can use the **Elastic Database client library** to create and manage scaled-out databases. This feature lets you easily develop sharded applications using hundreds—or even thousands—of Azure SQL databases.

To download:

- The Java version of the library, see [Maven Central Repository](#).
- The .NET version of the library, see [NuGet](#).

## Documentation

1. [Get started with Elastic Database tools](#)
2. [Elastic Database features](#)
3. [Shard map management](#)
4. [Migrate existing databases to scale out](#)
5. [Data dependent routing](#)
6. [Multi-shard queries](#)
7. [Adding a shard using Elastic Database tools](#)
8. [Multi-tenant applications with elastic database tools and row-level security](#)
9. [Upgrade client library apps](#)
10. [Elastic queries overview](#)
11. [Elastic database tools glossary](#)
12. [Elastic Database client library with Entity Framework](#)
13. [Elastic database client library with Dapper](#)
14. [Split-merge tool](#)
15. [Performance counters for shard map manager](#)
16. [FAQ for Elastic database tools](#)

## Client capabilities

Scaling out applications using *sharding* presents challenges for both the developer as well as the administrator. The client library simplifies the management tasks by providing tools that let both developers and administrators manage scaled-out databases. In a typical example, there are many databases, known as "shards," to manage. Customers are co-located in the same database, and there is one database per customer (a single-tenant scheme). The client library includes these features:

- **Shard Map Management:** A special database called the "shard map manager" is created. Shard map management is the ability for an application to manage metadata about its shards. Developers can use this functionality to register databases as shards, describe mappings of individual sharding keys or key ranges to those databases, and maintain this metadata as the number and composition of databases evolves to reflect capacity changes. Without the elastic database client library, you would need to spend a lot of time writing the management code when implementing sharding. For details, see [Shard map management](#).
- **Data dependent routing:** Imagine a request coming into the application. Based on the sharding key

value of the request, the application needs to determine the correct database based on the key value. It then opens a connection to the database to process the request. Data dependent routing provides the ability to open connections with a single easy call into the shard map of the application. Data dependent routing was another area of infrastructure code that is now covered by functionality in the elastic database client library. For details, see [Data dependent routing](#).

- **Multi-shard queries (MSQ):** Multi-shard querying works when a request involves several (or all) shards. A multi-shard query executes the same T-SQL code on all shards or a set of shards. The results from the participating shards are merged into an overall result set using UNION ALL semantics. The functionality as exposed through the client library handles many tasks, including: connection management, thread management, fault handling, and intermediate results processing. MSQ can query up to hundreds of shards. For details, see [Multi-shard querying](#).

In general, customers using elastic database tools can expect to get full T-SQL functionality when submitting shard-local operations as opposed to cross-shard operations that have their own semantics.

## Next steps

- Elastic Database Client Library ([Java](#), [.NET](#)) - to **download** the library.
- [Get started with elastic database tools](#) - to try the **sample app** that demonstrates client functions.
- GitHub ([Java](#), [.NET](#)) - to make contributions to the code.
- [Azure SQL Database elastic query overview](#) - to use elastic queries.
- [Moving data between scaled-out cloud databases](#) - for instructions on using the **split-merge tool**.

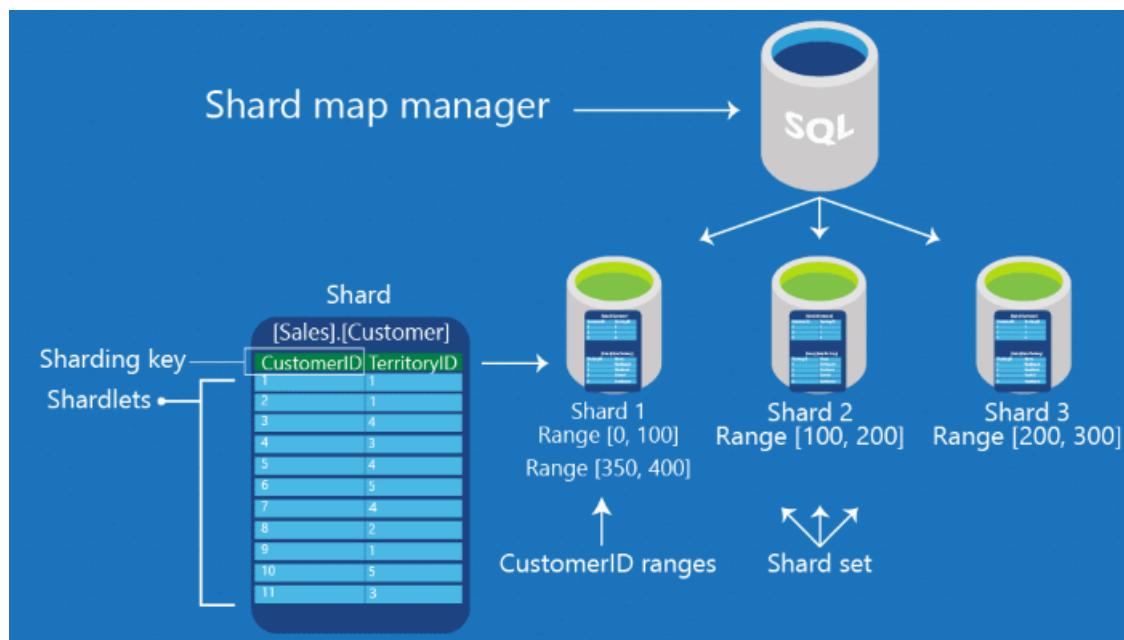
## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Scale out databases with the shard map manager

11/7/2019 • 12 minutes to read • [Edit Online](#)

To easily scale out databases on SQL Azure, use a shard map manager. The shard map manager is a special database that maintains global mapping information about all shards (databases) in a shard set. The metadata allows an application to connect to the correct database based upon the value of the **sharding key**. In addition, every shard in the set contains maps that track the local shard data (known as **shardlets**).



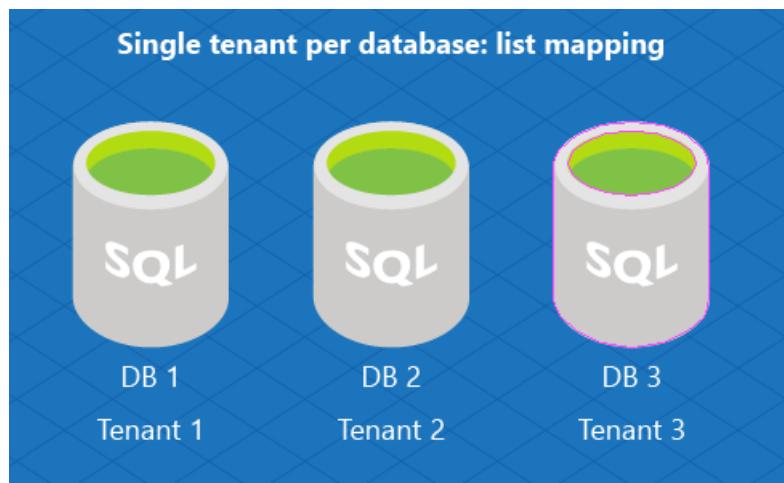
Understanding how these maps are constructed is essential to shard map management. This is done using the `ShardMapManager` class ([Java](#), [.NET](#), found in the [Elastic Database client library](#)) to manage shard maps.

## Shard maps and shard mappings

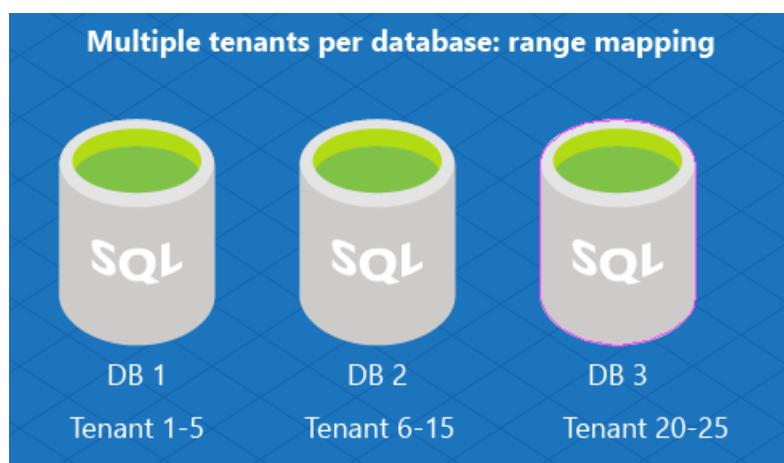
For each shard, you must select the type of shard map to create. The choice depends on the database architecture:

1. Single tenant per database
2. Multiple tenants per database (two types):
  - a. List mapping
  - b. Range mapping

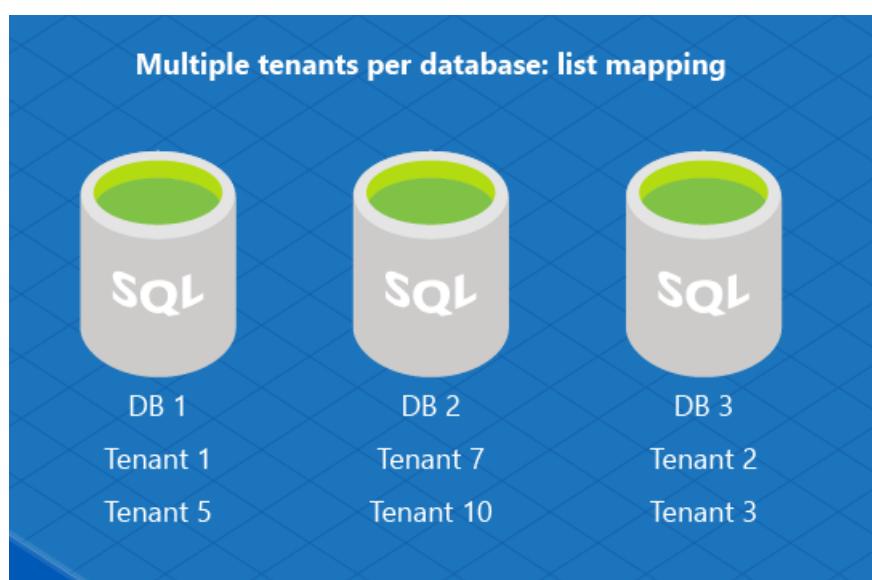
For a single-tenant model, create a **list-mapping** shard map. The single-tenant model assigns one database per tenant. This is an effective model for SaaS developers as it simplifies management.



The multi-tenant model assigns several tenants to an individual database (and you can distribute groups of tenants across multiple databases). Use this model when you expect each tenant to have small data needs. In this model, assign a range of tenants to a database using **range mapping**.



Or you can implement a multi-tenant database model using a *list mapping* to assign multiple tenants to an individual database. For example, DB1 is used to store information about tenant ID 1 and 5, and DB2 stores data for tenant 7 and tenant 10.



### Supported types for sharding keys

Elastic Scale support the following types as sharding keys:

.NET	JAVA
integer	integer
long	long
guid	uuid
byte[]	byte[]
datetime	timestamp
timespan	duration
datetimeoffset	offsetdatetime

### List and range shard maps

Shard maps can be constructed using **lists of individual sharding key values**, or they can be constructed using **ranges of sharding key values**.

#### List shard maps

**Shards** contain **shardlets** and the mapping of shardlets to shards is maintained by a shard map. A **list shard map** is an association between the individual key values that identify the shardlets and the databases that serve as shards. **List mappings** are explicit and different key values can be mapped to the same database. For example, key value 1 maps to Database A, and key values 3 and 6 both maps to Database B.

KEY	SHARD LOCATION
1	Database_A
3	Database_B
4	Database_C
6	Database_B
...	...

#### Range shard maps

In a **range shard map**, the key range is described by a pair **(Low Value, High Value)** where the *Low Value* is the minimum key in the range, and the *High Value* is the first value higher than the range.

For example, **[0, 100]** includes all integers greater than or equal 0 and less than 100. Note that multiple ranges can point to the same database, and disjoint ranges are supported (for example, [100,200) and [400,600) both point to Database C in the following example.)

KEY	SHARD LOCATION
[1,50)	Database_A
[50,100)	Database_B

KEY	SHARD LOCATION
[100,200)	Database_C
[400,600)	Database_C
...	...

Each of the tables shown above is a conceptual example of a **ShardMap** object. Each row is a simplified example of an individual **PointMapping** (for the list shard map) or **RangeMapping** (for the range shard map) object.

## Shard map manager

In the client library, the shard map manager is a collection of shard maps. The data managed by a **ShardMapManager** instance is kept in three places:

1. **Global Shard Map (GSM)**: You specify a database to serve as the repository for all of its shard maps and mappings. Special tables and stored procedures are automatically created to manage the information. This is typically a small database and lightly accessed, and it should not be used for other needs of the application. The tables are in a special schema named **\_\_ShardManagement**.
2. **Local Shard Map (LSM)**: Every database that you specify to be a shard is modified to contain several small tables and special stored procedures that contain and manage shard map information specific to that shard. This information is redundant with the information in the GSM, and it allows the application to validate cached shard map information without placing any load on the GSM; the application uses the LSM to determine if a cached mapping is still valid. The tables corresponding to the LSM on each shard are also in the schema **\_\_ShardManagement**.
3. **Application cache**: Each application instance accessing a **ShardMapManager** object maintains a local in-memory cache of its mappings. It stores routing information that has recently been retrieved.

## Constructing a ShardMapManager

A **ShardMapManager** object is constructed using a factory ([Java](#), [.NET](#)) pattern. The **ShardMapManagerFactory.GetSqlShardMapManager** ([Java](#), [.NET](#)) method takes credentials (including the server name and database name holding the GSM) in the form of a **ConnectionString** and returns an instance of a **ShardMapManager**.

**Please Note:** The **ShardMapManager** should be instantiated only once per app domain, within the initialization code for an application. Creation of additional instances of **ShardMapManager** in the same app domain results in increased memory and CPU utilization of the application. A **ShardMapManager** can contain any number of shard maps. While a single shard map may be sufficient for many applications, there are times when different sets of databases are used for different schema or for unique purposes; in those cases multiple shard maps may be preferable.

In this code, an application tries to open an existing **ShardMapManager** with the **TryGetSqlShardMapManager** ([Java](#), [.NET](#)) method. If objects representing a Global **ShardMapManager** (GSM) do not yet exist inside the database, the client library creates them using the **CreateSqlShardMapManager** ([Java](#), [.NET](#)) method.

```

// Try to get a reference to the Shard Map Manager in the shardMapManager database.
// If it doesn't already exist, then create it.
ShardMapManager shardMapManager = null;
boolean shardMapManagerExists =
ShardMapManagerFactory.tryGetSqlShardMapManager(shardMapManagerConnectionString,ShardMapManagerLoadPolicy.Lazy,
zy, refShardMapManager);
shardMapManager = refShardMapManager.argValue;

if (shardMapManagerExists) {
 ConsoleUtils.writeInfo("Shard Map %s already exists", shardMapManager);
}
else {
 // The Shard Map Manager does not exist, so create it
 shardMapManager = ShardMapManagerFactory.createSqlShardMapManager(shardMapManagerConnectionString);
 ConsoleUtils.writeInfo("Created Shard Map %s", shardMapManager);
}

```

```

// Try to get a reference to the Shard Map Manager via the Shard Map Manager database.
// If it doesn't already exist, then create it.
ShardMapManager shardMapManager;
bool shardMapManagerExists = ShardMapManagerFactory.TryGetSqlShardMapManager(
 connectionString,
 ShardMapManagerLoadPolicy.Lazy,
 out shardMapManager);

if (shardMapManagerExists)
{
 Console.WriteLine("Shard Map Manager already exists");
}
else
{
 // Create the Shard Map Manager.
 ShardMapManagerFactory.CreateSqlShardMapManager(connectionString);
 Console.WriteLine("Created SqlShardMapManager");

 shardMapManager = ShardMapManagerFactory.GetSqlShardMapManager(
 connectionString,
 ShardMapManagerLoadPolicy.Lazy);

 // The connectionString contains server name, database name, and admin credentials for privileges on both
 // the GSM and the shards themselves.
}

```

For the .NET version, you can use PowerShell to create a new Shard Map Manager. An example is available [here](#).

## Get a RangeShardMap or ListShardMap

After creating a shard map manager, you can get the RangeShardMap ([Java](#), [.NET](#)) or ListShardMap ([Java](#), [.NET](#)) using the TryGetRangeShardMap ([Java](#), [.NET](#)), the TryGetListShardMap ([Java](#), [.NET](#)), or the GetShardMap ([Java](#), [.NET](#)) method.

```

// Creates a new Range Shard Map with the specified name, or gets the Range Shard Map if it already exists.
static <T> RangeShardMap<T> createOrGetRangeShardMap(ShardMapManager shardMapManager,
 String shardMapName,
 ShardKeyType keyType) {
 // Try to get a reference to the Shard Map.
 ReferenceObjectHelper<RangeShardMap<T>> refRangeShardMap = new ReferenceObjectHelper<>(null);
 boolean isSuccess = shardMapManager.tryGetRangeShardMap(shardMapName, keyType, refRangeShardMap);
 RangeShardMap<T> shardMap = refRangeShardMap.returnValue;

 if (isSuccess && shardMap != null) {
 ConsoleUtils.writeInfo("Shard Map %1$s already exists", shardMap.getName());
 }
 else {
 // The Shard Map does not exist, so create it
 try {
 shardMap = shardMapManager.createRangeShardMap(shardMapName, keyType);
 }
 catch (Exception e) {
 e.printStackTrace();
 }
 ConsoleUtils.writeInfo("Created Shard Map %1$s", shardMap.getName());
 }

 return shardMap;
}

```

```

// Creates a new Range Shard Map with the specified name, or gets the Range Shard Map if it already exists.
public static RangeShardMap<T> CreateOrGetRangeShardMap<T>(ShardMapManager shardMapManager, string
shardMapName)
{
 // Try to get a reference to the Shard Map.
 RangeShardMap<T> shardMap;
 bool shardMapExists = shardMapManager.TryGetRangeShardMap(shardMapName, out shardMap);

 if (shardMapExists)
 {
 ConsoleUtils.WriteLine("Shard Map {0} already exists", shardMap.Name);
 }
 else
 {
 // The Shard Map does not exist, so create it
 shardMap = shardMapManager.CreateRangeShardMap<T>(shardMapName);
 ConsoleUtils.WriteLine("Created Shard Map {0}", shardMap.Name);
 }

 return shardMap;
}

```

## Shard map administration credentials

Applications that administer and manipulate shard maps are different from those that use the shard maps to route connections.

To administer shard maps (add or change shards, shard maps, shard mappings, etc.) you must instantiate the **ShardMapManager** using **credentials that have read/write privileges on both the GSM database and on each database that serves as a shard**. The credentials must allow for writes against the tables in both the GSM and LSM as shard map information is entered or changed, as well as for creating LSM tables on new shards.

See [Credentials used to access the Elastic Database client library](#).

### Only metadata affected

Methods used for populating or changing the **ShardMapManager** data do not alter the user data stored in the

shards themselves. For example, methods such as **CreateShard**, **DeleteShard**, **UpdateMapping**, etc. affect the shard map metadata only. They do not remove, add, or alter user data contained in the shards. Instead, these methods are designed to be used in conjunction with separate operations you perform to create or remove actual databases, or that move rows from one shard to another to rebalance a sharded environment. (The **split-merge** tool included with elastic database tools makes use of these APIs along with orchestrating actual data movement between shards.) See [Scaling using the Elastic Database split-merge tool](#).

## Data dependent routing

The shard map manager is used in applications that require database connections to perform the app-specific data operations. Those connections must be associated with the correct database. This is known as **Data Dependent Routing**.

For these applications, instantiate a shard map manager object from the factory using credentials that have read-only access on the GSM database. Individual requests for later connections supply credentials necessary for connecting to the appropriate shard database.

Note that these applications (using **ShardMapManager** opened with read-only credentials) cannot make changes to the maps or mappings. For those needs, create administrative-specific applications or PowerShell scripts that supply higher-privileged credentials as discussed earlier. See [Credentials used to access the Elastic Database client library](#).

For more information, see [Data dependent routing](#).

## Modifying a shard map

A shard map can be changed in different ways. All of the following methods modify the metadata describing the shards and their mappings, but they do not physically modify data within the shards, nor do they create or delete the actual databases. Some of the operations on the shard map described below may need to be coordinated with administrative actions that physically move data or that add and remove databases serving as shards.

These methods work together as the building blocks available for modifying the overall distribution of data in your sharded database environment.

- To add or remove shards: use **CreateShard** ([Java](#), [.NET](#)) and **DeleteShard** ([Java](#), [.NET](#)) of the shardmap ([Java](#), [.NET](#)) class.

The server and database representing the target shard must already exist for these operations to execute. These methods do not have any impact on the databases themselves, only on metadata in the shard map.

- To create or remove points or ranges that are mapped to the shards: use **CreateRangeMapping** ([Java](#), [.NET](#)), **DeleteMapping** ([Java](#), [.NET](#)) of the RangeShardMapping ([Java](#), [.NET](#)) class, and **CreatePointMapping** ([Java](#), [.NET](#)) of the ListShardMap ([Java](#), [.NET](#)) class.

Many different points or ranges can be mapped to the same shard. These methods only affect metadata - they do not affect any data that may already be present in shards. If data needs to be removed from the database in order to be consistent with **DeleteMapping** operations, you perform those operations separately but in conjunction with using these methods.

- To split existing ranges into two, or merge adjacent ranges into one: use **SplitMapping** ([Java](#), [.NET](#)) and **MergeMappings** ([Java](#), [.NET](#)).

Note that split and merge operations **do not change the shard to which key values are mapped**. A split breaks an existing range into two parts, but leaves both as mapped to the same shard. A merge operates on two adjacent ranges that are already mapped to the same shard, coalescing them into a single range. The movement of points or ranges themselves between shards needs to be coordinated by using **UpdateMapping** in conjunction with actual data movement. You can use the **Split/Merge** service

that is part of elastic database tools to coordinate shard map changes with data movement, when movement is needed.

- To re-map (or move) individual points or ranges to different shards: use **UpdateMapping** ([Java](#), [.NET](#)).

Since data may need to be moved from one shard to another in order to be consistent with **UpdateMapping** operations, you need to perform that movement separately but in conjunction with using these methods.

- To take mappings online and offline: use **MarkMappingOffline** ([Java](#), [.NET](#)) and **MarkMappingOnline** ([Java](#), [.NET](#)) to control the online state of a mapping.

Certain operations on shard mappings are only allowed when a mapping is in an “offline” state, including **UpdateMapping** and **DeleteMapping**. When a mapping is offline, a data-dependent request based on a key included in that mapping returns an error. In addition, when a range is first taken offline, all connections to the affected shard are automatically killed in order to prevent inconsistent or incomplete results for queries directed against ranges being changed.

Mappings are immutable objects in .Net. All of the methods above that change mappings also invalidate any references to them in your code. To make it easier to perform sequences of operations that change a mapping’s state, all of the methods that change a mapping return a new mapping reference, so operations can be chained. For example, to delete an existing mapping in shardmap sm that contains the key 25, you can execute the following:

```
sm.DeleteMapping(sm.MarkMappingOffline(sm.GetMappingForKey(25)));
```

## Adding a shard

Applications often need to add new shards to handle data that is expected from new keys or key ranges, for a shard map that already exists. For example, an application sharded by Tenant ID may need to provision a new shard for a new tenant, or data sharded monthly may need a new shard provisioned before the start of each new month.

If the new range of key values is not already part of an existing mapping and no data movement is necessary, it is simple to add the new shard and associate the new key or range to that shard. For details on adding new shards, see [Adding a new shard](#).

For scenarios that require data movement, however, the split-merge tool is needed to orchestrate the data movement between shards in combination with the necessary shard map updates. For details on using the split-merge tool, see [Overview of split-merge](#)

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Use Data-dependent routing to route a query to appropriate database

2/7/2020 • 6 minutes to read • [Edit Online](#)

**Data-dependent routing** is the ability to use the data in a query to route the request to an appropriate database. Data-dependant routing is a fundamental pattern when working with sharded databases. The request context may also be used to route the request, especially if the sharding key is not part of the query. Each specific query or transaction in an application using data-dependent routing is restricted to accessing one database per request. For the Azure SQL Database Elastic tools, this routing is accomplished with the **ShardMapManager** ([Java](#), [.NET](#)) class.

The application does not need to track various connection strings or DB locations associated with different slices of data in the sharded environment. Instead, the [Shard Map Manager](#) opens connections to the correct databases when needed, based on the data in the shard map and the value of the sharding key that is the target of the application's request. The key is typically the *customer\_id*, *tenant\_id*, *date\_key*, or some other specific identifier that is a fundamental parameter of the database request.

For more information, see [Scaling Out SQL Server with Data-Dependent Routing](#).

## Download the client library

To download:

- The Java version of the library, see [Maven Central Repository](#).
- The .NET version of the library, see [NuGet](#).

## Using a ShardMapManager in a data-dependent routing application

Applications should instantiate the **ShardMapManager** during initialization, using the factory call **GetSQLShardMapManager** ([Java](#), [.NET](#)). In this example, both a **ShardMapManager** and a specific **ShardMap** that it contains are initialized. This example shows the `GetSqlShardMapManager` and `GetRangeShardMap` ([Java](#), [.NET](#)) methods.

```
ShardMapManager smm = ShardMapManagerFactory.getSqlShardMapManager(connectionString,
ShardMapManagerLoadPolicy.Lazy);
RangeShardMap<int> rangeShardMap = smm.getRangeShardMap(Configuration.getRangeShardMapName(),
ShardKeyType.Int32);
```

```
ShardMapManager smm = ShardMapManagerFactory.GetSqlShardMapManager(smmConnectionString,
ShardMapManagerLoadPolicy.Lazy);
RangeShardMap<int> customerShardMap = smm.GetRangeShardMap<int>("customerMap");
```

### Use lowest privilege credentials possible for getting the shard map

If an application is not manipulating the shard map itself, the credentials used in the factory method should have read-only permissions on the **Global Shard Map** database. These credentials are typically different from credentials used to open connections to the shard map manager. See also [Credentials used to access the Elastic Database client library](#).

## Call the OpenConnectionForKey method

The **ShardMap.OpenConnectionForKey** method ([Java](#), [.NET](#)) returns a connection ready for issuing commands to the appropriate database based on the value of the **key** parameter. Shard information is cached in the application by the **ShardMapManager**, so these requests do not typically involve a database lookup against the **Global Shard Map** database.

```
// Syntax:
public Connection openConnectionForKey(Object key, String connectionString, ConnectionOptions options)
```

```
// Syntax:
public SqlConnection OpenConnectionForKey<TKey>(TKey key, string connectionString, ConnectionOptions
options)
```

- The **key** parameter is used as a lookup key into the shard map to determine the appropriate database for the request.
- The **connectionString** is used to pass only the user credentials for the desired connection. No database name or server name is included in this *connectionString* since the method determines the database and server using the **ShardMap**.
- The **connectionOptions** ([Java](#), [.NET](#)) should be set to **ConnectionOptions.Validate** if an environment where shard maps may change and rows may move to other databases as a result of split or merge operations. This validation involves a brief query to the local shard map on the target database (not to the global shard map) before the connection is delivered to the application.

If the validation against the local shard map fails (indicating that the cache is incorrect), the Shard Map Manager queries the global shard map to obtain the new correct value for the lookup, update the cache, and obtain and return the appropriate database connection.

Use **ConnectionOptions.None** only when shard mapping changes are not expected while an application is online. In that case, the cached values can be assumed to always be correct, and the extra round-trip validation call to the target database can be safely skipped. That reduces database traffic. The **connectionOptions** may also be set via a value in a configuration file to indicate whether sharding changes are expected or not during a period of time.

This example uses the value of an integer key **CustomerID**, using a **ShardMap** object named **customerShardMap**.

```
int customerId = 12345;
int productId = 4321;
// Looks up the key in the shard map and opens a connection to the shard
try (Connection conn = shardMap.openConnectionForKey(customerId,
Configuration.getCredentialsConnectionString())) {
 // Create a simple command that will insert or update the customer information
 PreparedStatement ps = conn.prepareStatement("UPDATE Sales.Customer SET PersonID = ? WHERE CustomerID = ?");

 ps.setInt(1, productId);
 ps.setInt(2, customerId);
 ps.executeUpdate();
} catch (SQLException e) {
 e.printStackTrace();
}
```

```

int customerId = 12345;
int newPersonId = 4321;

// Connect to the shard for that customer ID. No need to call a SqlConnection
// constructor followed by the Open method.
using (SqlConnection conn = customerShardMap.OpenConnectionForKey(customerId,
Configuration.GetCredentialsConnectionString(), ConnectionOptions.Validate))
{
 // Execute a simple command.
 SqlCommand cmd = conn.CreateCommand();
 cmd.CommandText = @"UPDATE Sales.Customer
 SET PersonID = @newPersonID WHERE CustomerID = @customerID";

 cmd.Parameters.AddWithValue("@customerID", customerId);cmd.Parameters.AddWithValue("@newPersonID",
newPersonId);
 cmd.ExecuteNonQuery();
}

```

The **OpenConnectionForKey** method returns a new already-open connection to the correct database. Connections utilized in this way still take full advantage of connection pooling.

The **OpenConnectionForKeyAsync** method ([Java](#), [.NET](#)) is also available if your application makes use of asynchronous programming.

## Integrating with transient fault handling

A best practice in developing data access applications in the cloud is to ensure that transient faults are caught by the app, and that the operations are retried several times before throwing an error. Transient fault handling for cloud applications is discussed at [Transient Fault Handling \(Java, .NET\)](#).

Transient fault handling can coexist naturally with the Data-Dependent Routing pattern. The key requirement is to retry the entire data access request including the **using** block that obtained the data-dependent routing connection. The preceding example could be rewritten as follows.

### Example - data-dependent routing with transient fault handling

```

int customerId = 12345;
int productId = 4321;
try {
 SqlDatabaseUtils.getSqlRetryPolicy().executeAction(() -> {
 // Looks up the key in the shard map and opens a connection to the shard
 try (Connection conn = shardMap.openConnectionForKey(customerId,
Configuration.getCredentialsConnectionString())) {
 // Create a simple command that will insert or update the customer information
 PreparedStatement ps = conn.prepareStatement("UPDATE Sales.Customer SET PersonID = ? WHERE
CustomerID = ?");
 ps.setInt(1, productId);
 ps.setInt(2, customerId);
 ps.executeUpdate();
 } catch (SQLException e) {
 e.printStackTrace();
 }
 });
} catch (Exception e) {
 throw new StoreException(e.getMessage(), e);
}

```

```

int customerId = 12345;
int newPersonId = 4321;

Configuration.SqlRetryPolicy.ExecuteAction(() -> {

 // Connect to the shard for a customer ID.
 using (SqlConnection conn = customerShardMap.OpenConnectionForKey(customerId,
Configuration.GetCredentialsConnectionString(), ConnectionOptions.Validate))
 {
 // Execute a simple command
 SqlCommand cmd = conn.CreateCommand();

 cmd.CommandText = @"UPDATE Sales.Customer
 SET PersonID = @newPersonID
 WHERE CustomerID = @customerID";

 cmd.Parameters.AddWithValue("@customerID", customerId);
 cmd.Parameters.AddWithValue("@newPersonID", newPersonId);
 cmd.ExecuteNonQuery();

 Console.WriteLine("Update completed");
 }
});

```

Packages necessary to implement transient fault handling are downloaded automatically when you build the elastic database sample application.

## Transactional consistency

Transactional properties are guaranteed for all operations local to a shard. For example, transactions submitted through data-dependent routing execute within the scope of the target shard for the connection. At this time, there are no capabilities provided for enlisting multiple connections into a transaction, and therefore there are no transactional guarantees for operations performed across shards.

## Next steps

To detach a shard, or to reattach a shard, see [Using the RecoveryManager class to fix shard map problems](#)

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Credentials used to access the Elastic Database client library

11/7/2019 • 3 minutes to read • [Edit Online](#)

The [Elastic Database client library](#) uses three different kinds of credentials to access the [shard map manager](#). Depending on the need, use the credential with the lowest level of access possible.

- **Management credentials:** for creating or manipulating a shard map manager. (See the [glossary](#).)
- **Access credentials:** to access an existing shard map manager to obtain information about shards.
- **Connection credentials:** to connect to shards.

See also [Managing databases and logins in Azure SQL Database](#).

## About management credentials

Management credentials are used to create a **ShardMapManager** ([Java](#), [.NET](#)) object for applications that manipulate shard maps. (For example, see [Adding a shard using Elastic Database tools](#) and [data-dependent routing](#)). The user of the elastic scale client library creates the SQL users and SQL logins and makes sure each is granted the read/write permissions on the global shard map database and all shard databases as well. These credentials are used to maintain the global shard map and the local shard maps when changes to the shard map are performed. For instance, use the management credentials to create the shard map manager object (using **GetSqlShardMapManager** ([Java](#), [.NET](#))):

```
// Obtain a shard map manager.
ShardMapManager shardMapManager =
 ShardMapManagerFactory.GetSqlShardMapManager(smmAdminConnectionString, ShardMapManagerLoadPolicy.Lazy);
```

The variable **smmAdminConnectionString** is a connection string that contains the management credentials. The user ID and password provide read/write access to both shard map database and individual shards. The management connection string also includes the server name and database name to identify the global shard map database. Here is a typical connection string for that purpose:

```
"Server=<yourserver>.database.windows.net;Database=<yourdatabase>;User ID=<yourmgmtusername>;Password=<yourmgmtpassword>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30;"
```

Do not use values in the form of "username@server"—instead just use the "username" value. This is because credentials must work against both the shard map manager database and individual shards, which may be on different servers.

## Access credentials

When creating a shard map manager in an application that does not administer shard maps, use credentials that have read-only permissions on the global shard map. The information retrieved from the global shard map under these credentials is used for [data-dependent routing](#) and to populate the shard map cache on the client. The credentials are provided through the same call pattern to **GetSqlShardMapManager**:

```
// Obtain shard map manager.
ShardMapManager shardMapManager = ShardMapManagerFactory.GetSqlShardMapManager(smmReadOnlyConnectionString,
ShardMapManagerLoadPolicy.Lazy);
```

Note the use of the **smmReadOnlyConnectionString** to reflect the use of different credentials for this access on behalf of **non-admin** users: these credentials should not provide write permissions on the global shard map.

## Connection credentials

Additional credentials are needed when using the **OpenConnectionForKey** ([Java](#), [.NET](#)) method to access a shard associated with a sharding key. These credentials need to provide permissions for read-only access to the local shard map tables residing on the shard. This is needed to perform connection validation for data-dependent routing on the shard. This code snippet allows data access in the context of data-dependent routing:

```
using (SqlConnection conn = rangeMap.OpenConnectionForKey<int>(targetWarehouse, smmUserConnectionString,
ConnectionOptions.Validate))
```

In this example, **smmUserConnectionString** holds the connection string for the user credentials. For Azure SQL DB, here is a typical connection string for user credentials:

```
"User ID=<yourusername>; Password=<youruserpassword>; Trusted_Connection=False; Encrypt=True; Connection
Timeout=30;"
```

As with the admin credentials, do not use values in the form of "username@server". Instead, just use "username". Also note that the connection string does not contain a server name and database name. That is because the **OpenConnectionForKey** call automatically directs the connection to the correct shard based on the key. Hence, the database name and server name are not provided.

## See also

[Managing databases and logins in Azure SQL Database](#)

[Securing your SQL Database](#)

[Elastic Database jobs](#)

## Additional resources

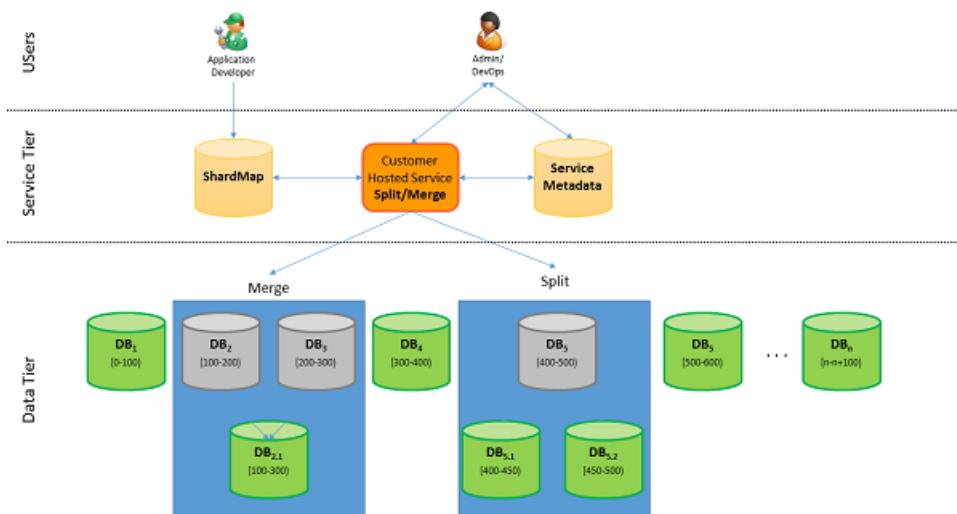
Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Moving data between scaled-out cloud databases

11/22/2019 • 18 minutes to read • [Edit Online](#)

If you are a Software as a Service developer, and suddenly your app undergoes tremendous demand, you need to accommodate the growth. So you add more databases (shards). How do you redistribute the data to the new databases without disrupting the data integrity? Use the **split-merge tool** to move data from constrained databases to the new databases.

The split-merge tool runs as an Azure web service. An administrator or developer uses the tool to move shardlets (data from a shard) between different databases (shards). The tool uses shard map management to maintain the service metadata database, and ensure consistent mappings.



## Download

[Microsoft.Azure.SqlDatabase.ElasticScale.Service.SplitMerge](#)

## Documentation

1. [Elastic database Split-Merge tool tutorial](#)
2. [Split-Merge security configuration](#)
3. [Split-merge security considerations](#)
4. [Shard map management](#)
5. [Migrate existing databases to scale-out](#)
6. [Elastic database tools](#)
7. [Elastic Database tools glossary](#)

## Why use the split-merge tool

- **Flexibility**

Applications need to stretch flexibly beyond the limits of a single Azure SQL DB database. Use the tool to move data as needed to new databases while retaining integrity.

- **Split to grow**

To increase overall capacity to handle explosive growth, create additional capacity by sharding the data

and by distributing it across incrementally more databases until capacity needs are fulfilled. This is a prime example of the **split** feature.

- **Merge to shrink**

Capacity needs shrink due to the seasonal nature of a business. The tool lets you scale down to fewer scale units when business slows. The 'merge' feature in the Elastic Scale split-merge Service covers this requirement.

- **Manage hotspots by moving shardlets**

With multiple tenants per database, the allocation of shardlets to shards can lead to capacity bottlenecks on some shards. This requires re-allocating shardlets or moving busy shardlets to new or less utilized shards.

## Concepts & key features

- **Customer-hosted services**

The split-merge is delivered as a customer-hosted service. You must deploy and host the service in your Microsoft Azure subscription. The package you download from NuGet contains a configuration template to complete with the information for your specific deployment. See the [split-merge tutorial](#) for details. Since the service runs in your Azure subscription, you can control and configure most security aspects of the service. The default template includes the options to configure SSL, certificate-based client authentication, encryption for stored credentials, DoS guarding and IP restrictions. You can find more information on the security aspects in the following document [split-merge security configuration](#).

The default deployed service runs with one worker and one web role. Each uses the A1 VM size in Azure Cloud Services. While you cannot modify these settings when deploying the package, you could change them after a successful deployment in the running cloud service, (through the Azure portal). Note that the worker role must not be configured for more than a single instance for technical reasons.

- **Shard map integration**

The split-merge service interacts with the shard map of the application. When using the split-merge service to split or merge ranges or to move shardlets between shards, the service automatically keeps the shard map up-to-date. To do so, the service connects to the shard map manager database of the application and maintains ranges and mappings as split/merge/move requests progress. This ensures that the shard map always presents an up-to-date view when split-merge operations are going on. Split, merge and shardlet movement operations are implemented by moving a batch of shardlets from the source shard to the target shard. During the shardlet movement operation the shardlets subject to the current batch are marked as offline in the shard map and are unavailable for data-dependent routing connections using the **OpenConnectionForKey** API.

- **Consistent shardlet connections**

When data movement starts for a new batch of shardlets, any shard-map provided data-dependent routing connections to the shard storing the shardlet are killed and subsequent connections from the shard map APIs to the shardlets are blocked while the data movement is in progress in order to avoid inconsistencies. Connections to other shardlets on the same shard will also get killed, but will succeed again immediately on retry. Once the batch is moved, the shardlets are marked online again for the target shard and the source data is removed from the source shard. The service goes through these steps for every batch until all shardlets have been moved. This will lead to several connection kill operations during the course of the complete split/merge/move operation.

- **Managing shardlet availability**

Limiting the connection killing to the current batch of shardlets as discussed above restricts the scope of unavailability to one batch of shardlets at a time. This is preferred over an approach where the complete shard would remain offline for all its shardlets during the course of a split or merge operation. The size of a batch, defined as the number of distinct shardlets to move at a time, is a configuration parameter. It can be defined for each split and merge operation depending on the application's availability and performance needs. Note that the range that is being locked in the shard map may be larger than the batch size specified. This is because the service picks the range size such that the actual number of sharding key values in the data approximately matches the batch size. This is important to remember in particular for sparsely populated sharding keys.

- **Metadata storage**

The split-merge service uses a database to maintain its status and to keep logs during request processing. The user creates this database in their subscription and provides the connection string for it in the configuration file for the service deployment. Administrators from the user's organization can also connect to this database to review request progress and to investigate detailed information regarding potential failures.

- **Sharding-awareness**

The split-merge service differentiates between (1) sharded tables, (2) reference tables, and (3) normal tables. The semantics of a split/merge/move operation depend on the type of the table used and are defined as follows:

- **Sharded tables**

Split, merge, and move operations move shardlets from source to target shard. After successful completion of the overall request, those shardlets are no longer present on the source. Note that the target tables need to exist on the target shard and must not contain data in the target range prior to processing of the operation.

- **Reference tables**

For reference tables, the split, merge and move operations copy the data from the source to the target shard. Note, however, that no changes occur on the target shard for a given table if any row is already present in this table on the target. The table has to be empty for any reference table copy operation to get processed.

- **Other Tables**

Other tables can be present on either the source or the target of a split and merge operation. The split-merge service disregards these tables for any data movement or copy operations. Note, however, that they can interfere with these operations in case of constraints.

The information on reference vs. sharded tables is provided by the `SchemaInfo` APIs on the shard map. The following example illustrates the use of these APIs on a given shard map manager object:

```

// Create the schema annotations
SchemaInfo schemaInfo = new SchemaInfo();

// reference tables
schemaInfo.Add(new ReferenceTableInfo("dbo", "region"));
schemaInfo.Add(new ReferenceTableInfo("dbo", "nation"));

// sharded tables
schemaInfo.Add(new ShardedTableInfo("dbo", "customer", "C_CUSTKEY"));
schemaInfo.Add(new ShardedTableInfo("dbo", "orders", "O_CUSTKEY"));

// publish
smm.GetSchemaInfoCollection().Add(Configuration.ShardMapName, schemaInfo);

```

The tables 'region' and 'nation' are defined as reference tables and will be copied with split/merge/move operations. 'customer' and 'orders' in turn are defined as sharded tables.

`C_CUSTKEY` and `O_CUSTKEY` serve as the sharding key.

- **Referential Integrity**

The split-merge service analyzes dependencies between tables and uses foreign key-primary key relationships to stage the operations for moving reference tables and shardlets. In general, reference tables are copied first in dependency order, then shardlets are copied in order of their dependencies within each batch. This is necessary so that FK-PK constraints on the target shard are honored as the new data arrives.

- **Shard Map Consistency and Eventual Completion**

In the presence of failures, the split-merge service resumes operations after any outage and aims to complete any in progress requests. However, there may be unrecoverable situations, e.g., when the target shard is lost or compromised beyond repair. Under those circumstances, some shardlets that were supposed to be moved may continue to reside on the source shard. The service ensures that shardlet mappings are only updated after the necessary data has been successfully copied to the target. Shardlets are only deleted on the source once all their data has been copied to the target and the corresponding mappings have been updated successfully. The deletion operation happens in the background while the range is already online on the target shard. The split-merge service always ensures correctness of the mappings stored in the shard map.

## The split-merge user interface

The split-merge service package includes a worker role and a web role. The web role is used to submit split-merge requests in an interactive way. The main components of the user interface are as follows:

- **Operation Type**

The operation type is a radio button that controls the kind of operation performed by the service for this request. You can choose between the split, merge and move scenarios. You can also cancel a previously submitted operation. You can use split, merge and move requests for range shard maps. List shard maps only support move operations.

- **Shard Map**

The next section of request parameters covers information about the shard map and the database hosting your shard map. In particular, you need to provide the name of the Azure SQL Database server and database hosting the shardmap, credentials to connect to the shard map database, and finally the name of the shard map. Currently, the operation only accepts a single set of credentials. These credentials need to have sufficient permissions to perform changes to the shard map as well as to the user data on the shards.

- **Source Range (split and merge)**

A split and merge operation processes a range using its low and high key. To specify an operation with an unbounded high key value, check the "High key is max" check box and leave the high key field empty. The range key values that you specify do not need to precisely match a mapping and its boundaries in your shard map. If you do not specify any range boundaries at all the service will infer the closest range for you automatically. You can use the GetMappings.ps1 PowerShell script to retrieve the current mappings in a given shard map.

- **Split Source Behavior (split)**

For split operations, define the point to split the source range. You do this by providing the sharding key where you want the split to occur. Use the radio button specify whether you want the lower part of the range (excluding the split key) to move, or whether you want the upper part to move (including the split key).

- **Source Shardlet (move)**

Move operations are different from split or merge operations as they do not require a range to describe the source. A source for move is simply identified by the sharding key value that you plan to move.

- **Target Shard (split)**

Once you have provided the information on the source of your split operation, you need to define where you want the data to be copied to by providing the Azure SQL Db server and database name for the target.

- **Target Range (merge)**

Merge operations move shardlets to an existing shard. You identify the existing shard by providing the range boundaries of the existing range that you want to merge with.

- **Batch Size**

The batch size controls the number of shardlets that will go offline at a time during the data movement. This is an integer value where you can use smaller values when you are sensitive to long periods of downtime for shardlets. Larger values will increase the time that a given shardlet is offline but may improve performance.

- **Operation ID (Cancel)**

If you have an ongoing operation that is no longer needed, you can cancel the operation by providing its operation ID in this field. You can retrieve the operation ID from the request status table (see Section 8.1) or from the output in the web browser where you submitted the request.

## Requirements and Limitations

The current implementation of the split-merge service is subject to the following requirements and limitations:

- The shards need to exist and be registered in the shard map before a split-merge operation on these shards can be performed.
- The service does not create tables or any other database objects automatically as part of its operations. This means that the schema for all sharded tables and reference tables needs to exist on the target shard prior to any split/merge/move operation. Sharded tables in particular are required to be empty in the range where new shardlets are to be added by a split/merge/move operation. Otherwise, the operation will fail the initial consistency check on the target shard. Also note that reference data is only copied if the reference table is empty and that there are no consistency guarantees with regard to other concurrent write operations on the reference tables. We recommend this: when running split/merge operations, no other write operations make

changes to the reference tables.

- The service relies on row identity established by a unique index or key that includes the sharding key to improve performance and reliability for large shardlets. This allows the service to move data at an even finer granularity than just the sharding key value. This helps to reduce the maximum amount of log space and locks that are required during the operation. Consider creating a unique index or a primary key including the sharding key on a given table if you want to use that table with split/merge/move requests. For performance reasons, the sharding key should be the leading column in the key or the index.
- During the course of request processing, some shardlet data may be present both on the source and the target shard. This is necessary to protect against failures during the shardlet movement. The integration of split-merge with the shard map ensures that connections through the data-dependent routing APIs using the **OpenConnectionForKey** method on the shard map do not see any inconsistent intermediate states. However, when connecting to the source or the target shards without using the **OpenConnectionForKey** method, inconsistent intermediate states might be visible when split/merge/move requests are going on. These connections may show partial or duplicate results depending on the timing or the shard underlying the connection. This limitation currently includes the connections made by Elastic Scale Multi-Shard-Queries.
- The metadata database for the split-merge service must not be shared between different roles. For example, a role of the split-merge service running in staging needs to point to a different metadata database than the production role.

## Billing

The split-merge service runs as a cloud service in your Microsoft Azure subscription. Therefore charges for cloud services apply to your instance of the service. Unless you frequently perform split/merge/move operations, we recommend you delete your split-merge cloud service. That saves costs for running or deployed cloud service instances. You can re-deploy and start your readily runnable configuration whenever you need to perform split or merge operations.

## Monitoring

### Status tables

The split-merge Service provides the **RequestStatus** table in the metadata store database for monitoring of completed and ongoing requests. The table lists a row for each split-merge request that has been submitted to this instance of the split-merge service. It gives the following information for each request:

- **Timestamp**

The time and date when the request was started.

- **OperationId**

A GUID that uniquely identifies the request. This request can also be used to cancel the operation while it is still ongoing.

- **Status**

The current state of the request. For ongoing requests, it also lists the current phase in which the request is.

- **CancelRequest**

A flag that indicates whether the request has been canceled.

- **Progress**

A percentage estimate of completion for the operation. A value of 50 indicates that the operation is approximately 50% complete.

- **Details**

An XML value that provides a more detailed progress report. The progress report is periodically updated as sets of rows are copied from source to target. In case of failures or exceptions, this column also includes more detailed information about the failure.

## Azure Diagnostics

The split-merge service uses Azure Diagnostics based on Azure SDK 2.5 for monitoring and diagnostics. You control the diagnostics configuration as explained here: [Enabling Diagnostics in Azure Cloud Services and Virtual Machines](#). The download package includes two diagnostics configurations - one for the web role and one for the worker role. It includes the definitions to log Performance Counters, IIS logs, Windows Event Logs, and split-merge application event logs.

## Deploy Diagnostics

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

To enable monitoring and diagnostics using the diagnostic configuration for the web and worker roles provided by the NuGet package, run the following commands using Azure PowerShell:

```
$storageName = "<azureStorageAccount>"
$key = "<azureStorageAccountKey>"
$storageContext = New-AzStorageContext -StorageAccountName $storageName -StorageAccountKey $key
$configPath = "<filePath>\SplitMergeWebContent.diagnostics.xml"
$serviceName = "<cloudServiceName>"

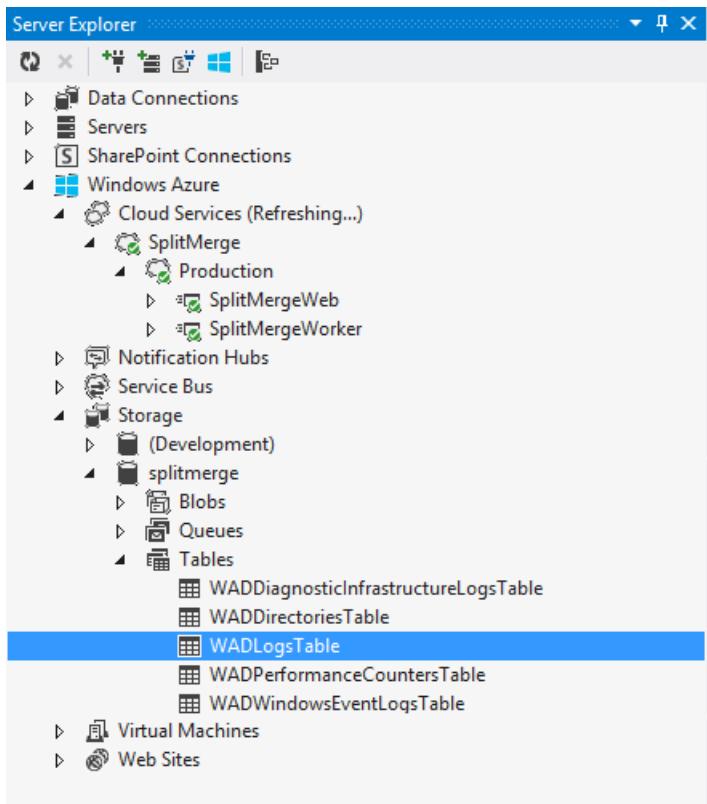
Set-AzureServiceDiagnosticsExtension -StorageContext $storageContext `
 -DiagnosticsConfigurationPath $configPath -ServiceName $serviceName `
 -Slot Production -Role "SplitMergeWeb"

Set-AzureServiceDiagnosticsExtension -StorageContext $storageContext `
 -DiagnosticsConfigurationPath $configPath -ServiceName $serviceName `
 -Slot Production -Role "SplitMergeWorker"
```

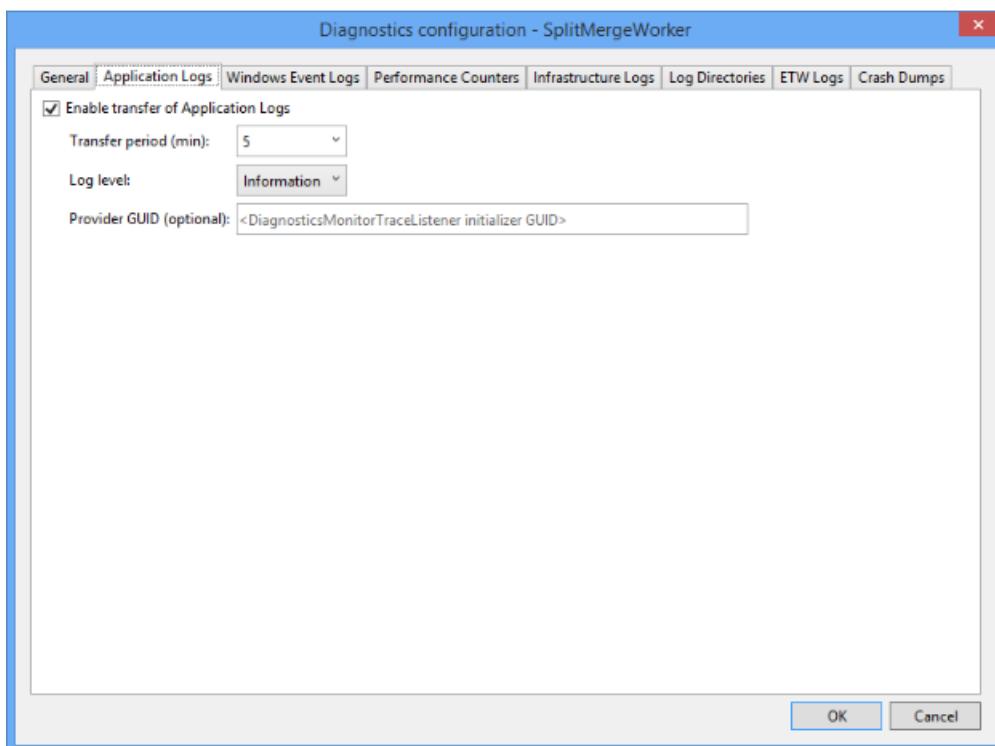
You can find more information on how to configure and deploy diagnostics settings here: [Enabling Diagnostics in Azure Cloud Services and Virtual Machines](#).

## Retrieve diagnostics

You can easily access your diagnostics from the Visual Studio Server Explorer in the Azure part of the Server Explorer tree. Open a Visual Studio instance, and in the menu bar click View, and Server Explorer. Click the Azure icon to connect to your Azure subscription. Then navigate to Azure -> Storage -> <your storage account> -> Tables -> WADLogsTable. For more information, see [Server Explorer](#).



The WADLogsTable highlighted in the figure above contains the detailed events from the split-merge service's application log. Note that the default configuration of the downloaded package is geared towards a production deployment. Therefore the interval at which logs and counters are pulled from the service instances is large (5 minutes). For test and development, lower the interval by adjusting the diagnostics settings of the web or the worker role to your needs. Right-click on the role in the Visual Studio Server Explorer (see above) and then adjust the Transfer Period in the dialog for the Diagnostics configuration settings:



## Performance

In general, better performance is to be expected from the higher, more performant service tiers in Azure SQL Database. Higher IO, CPU and memory allocations for the higher service tiers benefit the bulk copy and delete operations that the split-merge service uses. For that reason, increase the service tier just for those databases for

a defined, limited period of time.

The service also performs validation queries as part of its normal operations. These validation queries check for unexpected presence of data in the target range and ensure that any split/merge/move operation starts from a consistent state. These queries all work over sharding key ranges defined by the scope of the operation and the batch size provided as part of the request definition. These queries perform best when an index is present that has the sharding key as the leading column.

In addition, a uniqueness property with the sharding key as the leading column will allow the service to use an optimized approach that limits resource consumption in terms of log space and memory. This uniqueness property is required to move large data sizes (typically above 1GB).

## How to upgrade

1. Follow the steps in [Deploy a split-merge service](#).
2. Change your cloud service configuration file for your split-merge deployment to reflect the new configuration parameters. A new required parameter is the information about the certificate used for encryption. An easy way to do this is to compare the new configuration template file from the download against your existing configuration. Make sure you add the settings for "DataEncryptionPrimaryCertificateThumbprint" and "DataEncryptionPrimary" for both the web and the worker role.
3. Before deploying the update to Azure, ensure that all currently running split-merge operations have finished. You can easily do this by querying the RequestStatus and PendingWorkflows tables in the split-merge metadata database for ongoing requests.
4. Update your existing cloud service deployment for split-merge in your Azure subscription with the new package and your updated service configuration file.

You do not need to provision a new metadata database for split-merge to upgrade. The new version will automatically upgrade your existing metadata database to the new version.

## Best practices & troubleshooting

- Define a test tenant and exercise your most important split/merge/move operations with the test tenant across several shards. Ensure that all metadata is defined correctly in your shard map and that the operations do not violate constraints or foreign keys.
- Keep the test tenant data size above the maximum data size of your largest tenant to ensure you are not encountering data size related issues. This helps you assess an upper bound on the time it takes to move a single tenant around.
- Make sure that your schema allows deletions. The split-merge service requires the ability to remove data from the source shard once the data has been successfully copied to the target. For example, **delete triggers** can prevent the service from deleting the data on the source and may cause operations to fail.
- The sharding key should be the leading column in your primary key or unique index definition. That ensures the best performance for the split or merge validation queries, and for the actual data movement and deletion operations which always operate on sharding key ranges.
- Collocate your split-merge service in the region and data center where your databases reside.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Elastic database tools frequently asked questions (FAQ)

11/7/2019 • 2 minutes to read • [Edit Online](#)

## If I have a single-tenant per shard and no sharding key, how do I populate the sharding key for the schema info

The schema info object is only used to split merge scenarios. If an application is inherently single-tenant, then it does not require the Split Merge tool and thus there is no need to populate the schema info object.

## I've provisioned a database and I already have a Shard Map Manager, how do I register this new database as a shard

Please see [Adding a shard to an application using the elastic database client library](#).

## How much do elastic database tools cost

Using the elastic database client library does not incur any costs. Costs accrue only for the Azure SQL databases that you use for shards and the Shard Map Manager, as well as the web/worker roles you provision for the Split Merge tool.

## Why are my credentials not working when I add a shard from a different server

Do not use credentials in the form of "User ID=username@servername", instead simply use "User ID = username". Also, be sure that the "username" login has permissions on the shard.

## Do I need to create a Shard Map Manager and populate shards every time I start my applications

No—the creation of the Shard Map Manager (for example, `ShardMapManagerFactory.CreateSqlShardMapManager()`) is a one-time operation. Your application should use the call `ShardMapManagerFactory.TryGetSqlShardMapManager()` at application start-up time. There should only one such call per application domain.

## I have questions about using elastic database tools, how do I get them answered

Please reach out to us on the [SQL Database forum](#).

## When I get a database connection using a sharding key, I can still query data for other sharding keys on the same shard. Is this by design

The Elastic Scale APIs give you a connection to the correct database for your sharding key, but do not provide sharding key filtering. Add **WHERE** clauses to your query to restrict the scope to the provided sharding key, if necessary.

## Can I use a different SQL Database edition for each shard in my shard set

Yes, a shard is an individual database, and thus one shard could be a Premium edition while another be a Standard edition. Further, the edition of a shard can scale up or down multiple times during the lifetime of the shard.

## Does the Split Merge tool provision (or delete) a database during a split or merge operation

No. For **split** operations, the target database must exist with the appropriate schema and be registered with the Shard Map Manager. For **merge** operations, you must delete the shard from the shard map manager and then delete the database.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Azure SQL Database elastic query overview (preview)

12/5/2019 • 10 minutes to read • [Edit Online](#)

The elastic query feature (in preview) enables you to run a Transact-SQL query that spans multiple databases in Azure SQL Database. It allows you to perform cross-database queries to access remote tables, and to connect Microsoft and third-party tools (Excel, Power BI, Tableau, etc.) to query across data tiers with multiple databases. Using this feature, you can scale out queries to large data tiers in SQL Database and visualize the results in business intelligence (BI) reports.

## Why use elastic queries

### Azure SQL Database

Query across Azure SQL databases completely in T-SQL. This allows for read-only querying of remote databases and provides an option for current on-premises SQL Server customers to migrate applications using three- and four-part names or linked server to SQL DB.

### Available on standard tier

Elastic query is supported on both the Standard and Premium service tiers. See the section on Preview Limitations below on performance limitations for lower service tiers.

### Push parameters to remote databases

Elastic queries can now push SQL parameters to the remote databases for execution.

### Stored procedure execution

Execute remote stored procedure calls or remote functions using `sp_execute_remote`.

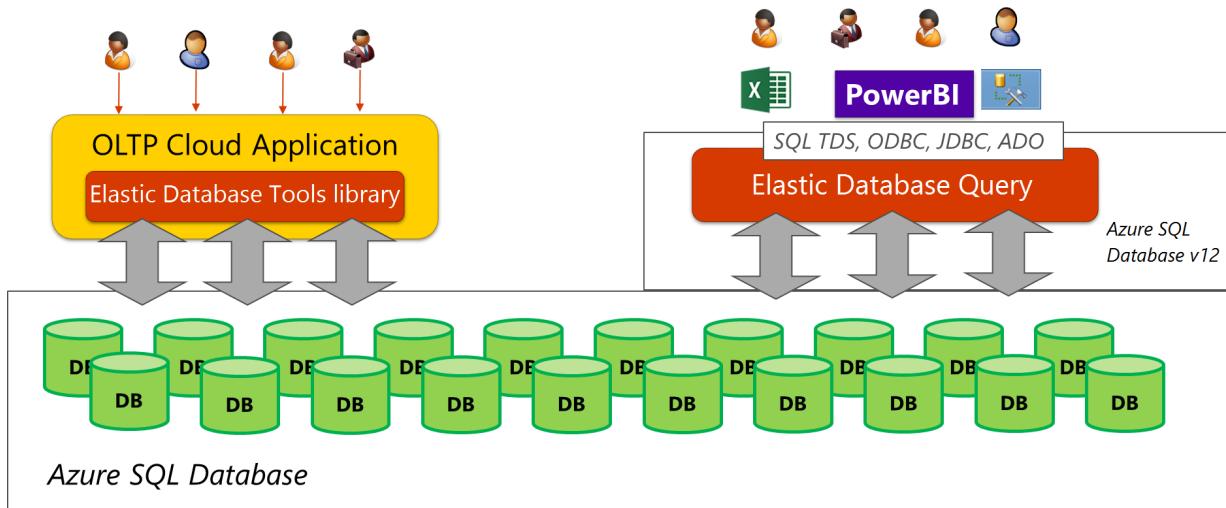
### Flexibility

External tables with elastic query can refer to remote tables with a different schema or table name.

## Elastic query scenarios

The goal is to facilitate querying scenarios where multiple databases contribute rows into a single overall result. The query can either be composed by the user or application directly, or indirectly through tools that are connected to the database. This is especially useful when creating reports, using commercial BI or data integration tools, or any application that cannot be changed. With an elastic query, you can query across several databases using the familiar SQL Server connectivity experience in tools such as Excel, Power BI, Tableau, or Cognos. An elastic query allows easy access to an entire collection of databases through queries issued by SQL Server Management Studio or Visual Studio, and facilitates cross-database querying from Entity Framework or other ORM environments. Figure 1 shows a scenario where an existing cloud application (which uses the [elastic database client library](#)) builds on a scaled-out data tier, and an elastic query is used for cross-database reporting.

**Figure 1** Elastic query used on scaled-out data tier



Customer scenarios for elastic query are characterized by the following topologies:

- **Vertical partitioning - Cross-database queries** (Topology 1): The data is partitioned vertically between a number of databases in a data tier. Typically, different sets of tables reside on different databases. That means that the schema is different on different databases. For instance, all tables for inventory are on one database while all accounting-related tables are on a second database. Common use cases with this topology require one to query across or to compile reports across tables in several databases.
- **Horizontal Partitioning - Sharding** (Topology 2): Data is partitioned horizontally to distribute rows across a scaled out data tier. With this approach, the schema is identical on all participating databases. This approach is also called "sharding". Sharding can be performed and managed using (1) the elastic database tools libraries or (2) self-sharding. An elastic query is used to query or compile reports across many shards. Shards are typically databases within an elastic pool. You can think of elastic query as an efficient way for querying all databases of elastic pool at once, as long as databases share the common schema.

#### NOTE

Elastic query works best for reporting scenarios where most of the processing (filtering, aggregation) can be performed on the external source side. It is not suitable for ETL operations where large amount of data is being transferred from remote database(s). For heavy reporting workloads or data warehousing scenarios with more complex queries, also consider using [Azure Synapse Analytics](#).

## Vertical partitioning - cross-database queries

To begin coding, see [Getting started with cross-database query \(vertical partitioning\)](#).

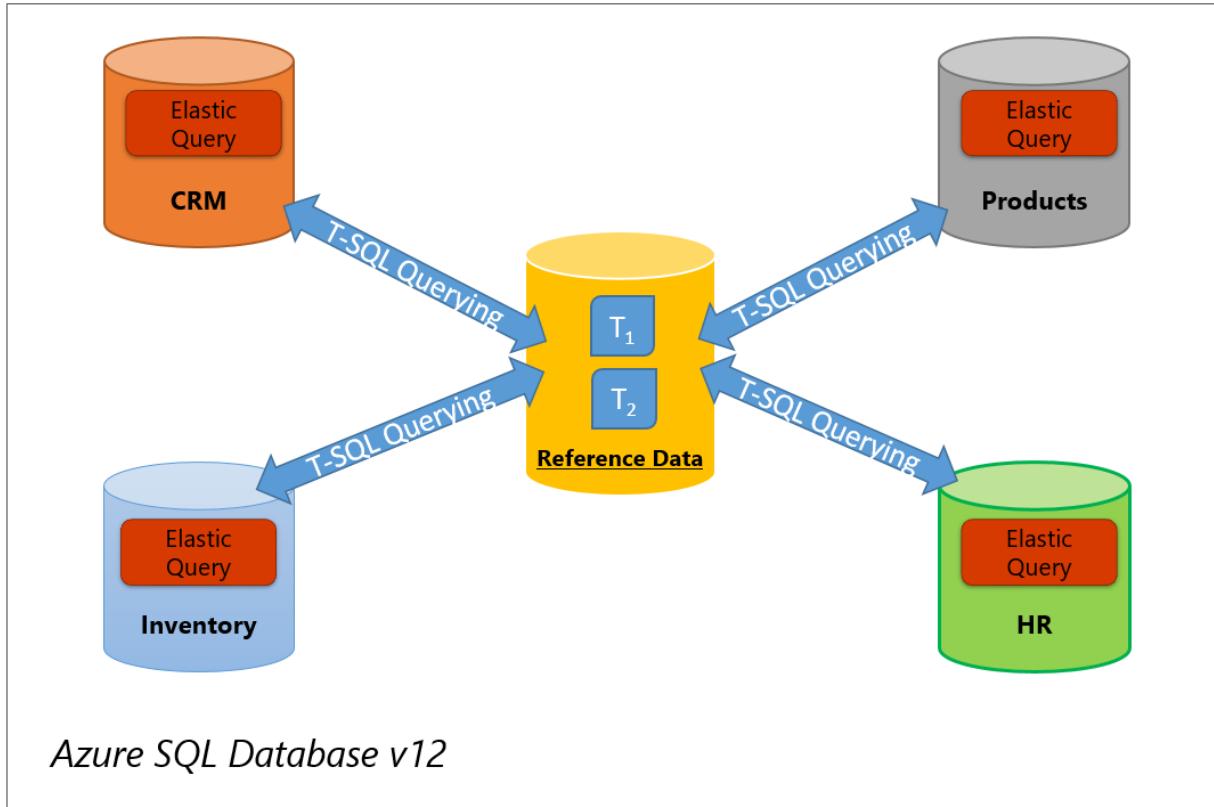
An elastic query can be used to make data located in a SQL database available to other SQL databases. This allows queries from one database to refer to tables in any other remote SQL database. The first step is to define an external data source for each remote database. The external data source is defined in the local database from which you want to gain access to tables located on the remote database. No changes are necessary on the remote database. For typical vertical partitioning scenarios where different databases have different schemas, elastic queries can be used to implement common use cases such as access to reference data and cross-database querying.

#### IMPORTANT

You must possess ALTER ANY EXTERNAL DATA SOURCE permission. This permission is included with the ALTER DATABASE permission. ALTER ANY EXTERNAL DATA SOURCE permissions are needed to refer to the underlying data source.

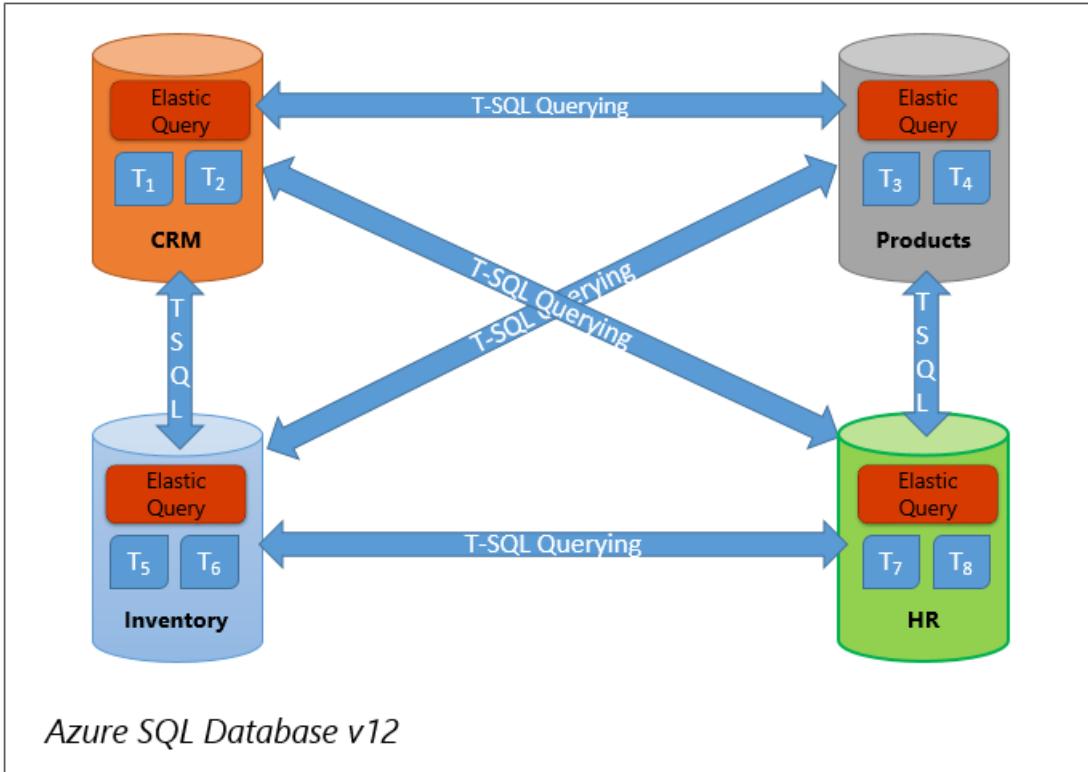
**Reference data:** The topology is used for reference data management. In the figure below, two tables (T1 and T2) with reference data are kept on a dedicated database. Using an elastic query, you can now access tables T1 and T2 remotely from other databases, as shown in the figure. Use topology 1 if reference tables are small or remote queries into reference table have selective predicates.

**Figure 2** Vertical partitioning - Using elastic query to query reference data



**Cross-database querying:** Elastic queries enable use cases that require querying across several SQL databases. Figure 3 shows four different databases: CRM, Inventory, HR, and Products. Queries performed in one of the databases also need access to one or all the other databases. Using an elastic query, you can configure your database for this case by running a few simple DDL statements on each of the four databases. After this one-time configuration, access to a remote table is as simple as referring to a local table from your T-SQL queries or from your BI tools. This approach is recommended if the remote queries do not return large results.

**Figure 3** Vertical partitioning - Using elastic query to query across various databases



The following steps configure elastic database queries for vertical partitioning scenarios that require access to a table located on remote SQL databases with the same schema:

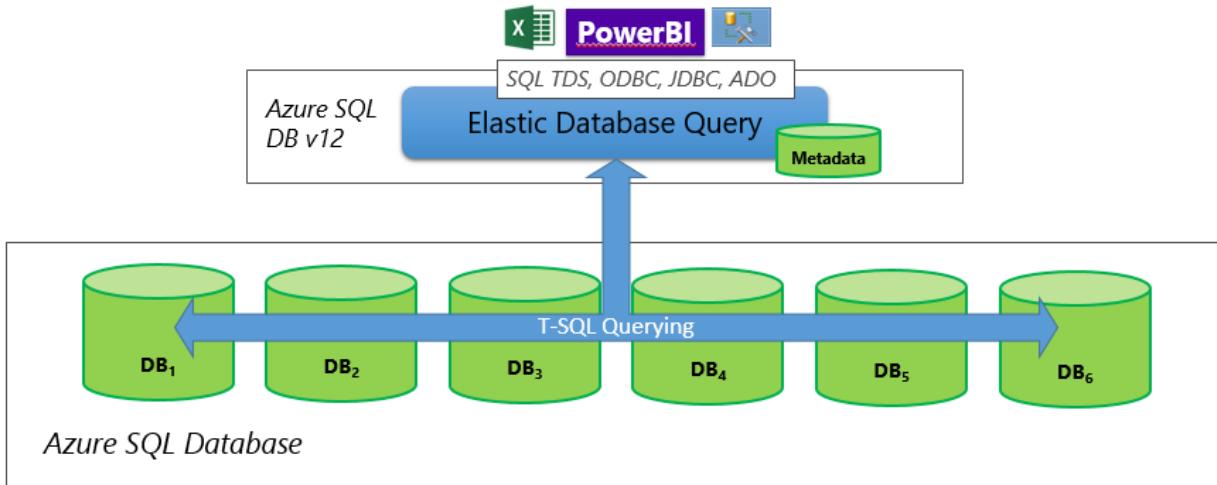
- `CREATE MASTER KEY` mymasterkey
- `CREATE DATABASE SCOPED CREDENTIAL` mycredential
- `CREATE/DROP EXTERNAL DATA SOURCE` mydatasource of type **RDBMS**
- `CREATE/DROP EXTERNAL TABLE` mytable

After running the DDL statements, you can access the remote table "mytable" as though it were a local table. Azure SQL Database automatically opens a connection to the remote database, processes your request on the remote database, and returns the results.

## Horizontal partitioning - sharding

Using elastic query to perform reporting tasks over a sharded, that is, horizontally partitioned, data tier requires an [elastic database shard map](#) to represent the databases of the data tier. Typically, only a single shard map is used in this scenario and a dedicated database with elastic query capabilities (head node) serves as the entry point for reporting queries. Only this dedicated database needs access to the shard map. Figure 4 illustrates this topology and its configuration with the elastic query database and shard map. The databases in the data tier can be of any Azure SQL Database version or edition. For more information about the elastic database client library and creating shard maps, see [Shard map management](#).

**Figure 4** Horizontal partitioning - Using elastic query for reporting over sharded data tiers



#### NOTE

Elastic Query Database (head node) can be separate database, or it can be the same database that hosts the shard map. Whatever configuration you choose, make sure that service tier and compute size of that database is high enough to handle the expected amount of login/query requests.

The following steps configure elastic database queries for horizontal partitioning scenarios that require access to a set of tables located on (typically) several remote SQL databases:

- `CREATE MASTER KEY` mymasterkey
- `CREATE DATABASE SCOPED CREDENTIAL` mycredential
- Create a `shard map` representing your data tier using the elastic database client library.
- `CREATE/DROP EXTERNAL DATA SOURCE` mydatasource of type **SHARD\_MAP\_MANAGER**
- `CREATE/DROP EXTERNAL TABLE` mytable

Once you have performed these steps, you can access the horizontally partitioned table "mytable" as though it were a local table. Azure SQL Database automatically opens multiple parallel connections to the remote databases where the tables are physically stored, processes the requests on the remote databases, and returns the results. More information on the steps required for the horizontal partitioning scenario can be found in [elastic query for horizontal partitioning](#).

To begin coding, see [Getting started with elastic query for horizontal partitioning \(sharding\)](#).

#### IMPORTANT

Successful execution of elastic query over a large set of databases relies heavily on the availability of each of databases during the query execution. If one of databases is not available, entire query will fail. If you plan to query hundreds or thousands of databases at once, make sure your client application has retry logic embedded, or consider leveraging [Elastic Database Jobs](#) (preview) and querying smaller subsets of databases, consolidating results of each query into a single destination.

## T-SQL querying

Once you have defined your external data sources and your external tables, you can use regular SQL Server connection strings to connect to the databases where you defined your external tables. You can then run T-SQL statements over your external tables on that connection with the limitations outlined below. You can find more information and examples of T-SQL queries in the documentation topics for [horizontal partitioning](#) and [vertical partitioning](#).

# Connectivity for tools

You can use regular SQL Server connection strings to connect your applications and BI or data integration tools to databases that have external tables. Make sure that SQL Server is supported as a data source for your tool. Once connected, refer to the elastic query database and the external tables in that database just like you would do with any other SQL Server database that you connect to with your tool.

## IMPORTANT

Authentication using Azure Active Directory with elastic queries is not currently supported.

# Cost

Elastic query is included into the cost of Azure SQL Database databases. Note that topologies where your remote databases are in a different data center than the elastic query endpoint are supported, but data egress from remote databases is charged regularly [Azure rates](#).

# Preview limitations

- Running your first elastic query can take up to a few minutes on the Standard service tier. This time is necessary to load the elastic query functionality; loading performance improves with higher service tiers and compute sizes.
- Scripting of external data sources or external tables from SSMS or SSDT is not yet supported.
- Import/Export for SQL DB does not yet support external data sources and external tables. If you need to use Import/Export, drop these objects before exporting and then re-create them after importing.
- Elastic query currently only supports read-only access to external tables. You can, however, use full T-SQL functionality on the database where the external table is defined. This can be useful to, e.g., persist temporary results using, for example, `SELECT <column_list> INTO <local_table>`, or to define stored procedures on the elastic query database that refer to external tables.
- Except for nvarchar(max), LOB types (including spatial types) are not supported in external table definitions. As a workaround, you can create a view on the remote database that casts the LOB type into nvarchar(max), define your external table over the view instead of the base table and then cast it back into the original LOB type in your queries.
- Columns of nvarchar(max) data type in result set disable advanced batching technics used in Elastic Query implementation and may affect performance of query for an order of magnitude, or even two orders of magnitude in non-canonical use cases where large amount of non-aggregated data is being transferred as a result of query.
- Column statistics over external tables are currently not supported. Table statistics are supported, but need to be created manually.
- Elastic query works with Azure SQL Database only. You cannot use it for querying on-premises SQL Server, or SQL Server in a VM.

# Feedback

Share feedback on your experience with elastic queries with us below, on the MSDN forums, or on Stack Overflow. We are interested in all kinds of feedback about the service (defects, rough edges, feature gaps).

# Next steps

- For a vertical partitioning tutorial, see [Getting started with cross-database query \(vertical partitioning\)](#).
- For syntax and sample queries for vertically partitioned data, see [Querying vertically partitioned data](#)

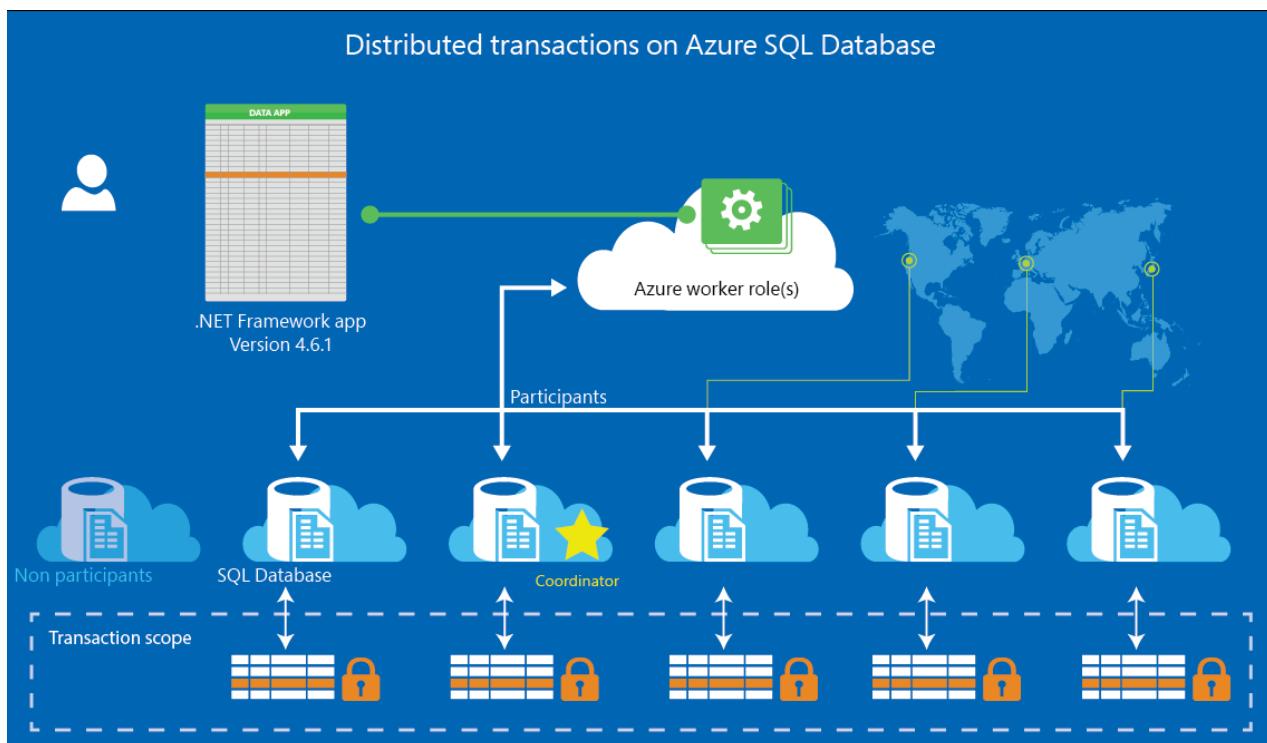
- For a horizontal partitioning (sharding) tutorial, see [Getting started with elastic query for horizontal partitioning \(sharding\)](#).
- For syntax and sample queries for horizontally partitioned data, see [Querying horizontally partitioned data](#)
- See [sp\\_execute\\_remote](#) for a stored procedure that executes a Transact-SQL statement on a single remote Azure SQL Database or set of databases serving as shards in a horizontal partitioning scheme.

# Distributed transactions across cloud databases

7/26/2019 • 8 minutes to read • [Edit Online](#)

Elastic database transactions for Azure SQL Database (SQL DB) allow you to run transactions that span several databases in SQL DB. Elastic database transactions for SQL DB are available for .NET applications using ADO .NET and integrate with the familiar programming experience using the `System.Transaction` classes. To get the library, see [.NET Framework 4.6.1 \(Web Installer\)](#).

On premises, such a scenario usually required running Microsoft Distributed Transaction Coordinator (MSDTC). Since MSDTC is not available for Platform-as-a-Service application in Azure, the ability to coordinate distributed transactions has now been directly integrated into SQL DB. Applications can connect to any SQL Database to launch distributed transactions, and one of the databases will transparently coordinate the distributed transaction, as shown in the following figure.



## Common scenarios

Elastic database transactions for SQL DB enable applications to make atomic changes to data stored in several different SQL Databases. The preview focuses on client-side development experiences in C# and .NET. A server-side experience using T-SQL is planned for a later time.

Elastic database transactions targets the following scenarios:

- Multi-database applications in Azure: With this scenario, data is vertically partitioned across several databases in SQL DB such that different kinds of data reside on different databases. Some operations require changes to data which is kept in two or more databases. The application uses elastic database transactions to coordinate the changes across databases and ensure atomicity.
- Sharded database applications in Azure: With this scenario, the data tier uses the [Elastic Database client library](#) or self-sharding to horizontally partition the data across many databases in SQL DB. One prominent use case is the need to perform atomic changes for a sharded multi-tenant application when changes span tenants. Think for instance of a transfer from one tenant to another, both residing on different databases. A second case is fine-grained sharding to accommodate capacity needs for a large tenant which in turn typically implies that

some atomic operations needs to stretch across several databases used for the same tenant. A third case is atomic updates to reference data that are replicated across databases. Atomic, transacted, operations along these lines can now be coordinated across several databases using the preview. Elastic database transactions use two-phase commit to ensure transaction atomicity across databases. It is a good fit for transactions that involve less than 100 databases at a time within a single transaction. These limits are not enforced, but one should expect performance and success rates for elastic database transactions to suffer when exceeding these limits.

## Installation and migration

The capabilities for elastic database transactions in SQL DB are provided through updates to the .NET libraries System.Data.dll and System.Transactions.dll. The DLLs ensure that two-phase commit is used where necessary to ensure atomicity. To start developing applications using elastic database transactions, install [.NET Framework 4.6.1](#) or a later version. When running on an earlier version of the .NET framework, transactions will fail to promote to a distributed transaction and an exception will be raised.

After installation, you can use the distributed transaction APIs in System.Transactions with connections to SQL DB. If you have existing MSDTC applications using these APIs, simply rebuild your existing applications for .NET 4.6 after installing the 4.6.1 Framework. If your projects target .NET 4.6, they will automatically use the updated DLLs from the new Framework version and distributed transaction API calls in combination with connections to SQL DB will now succeed.

Remember that elastic database transactions do not require installing MSDTC. Instead, elastic database transactions are directly managed by and within SQL DB. This significantly simplifies cloud scenarios since a deployment of MSDTC is not necessary to use distributed transactions with SQL DB. Section 4 explains in more detail how to deploy elastic database transactions and the required .NET framework together with your cloud applications to Azure.

## Development experience

### Multi-database applications

The following sample code uses the familiar programming experience with .NET System.Transactions. The TransactionScope class establishes an ambient transaction in .NET. (An “ambient transaction” is one that lives in the current thread.) All connections opened within the TransactionScope participate in the transaction. If different databases participate, the transaction is automatically elevated to a distributed transaction. The outcome of the transaction is controlled by setting the scope to complete to indicate a commit.

```
using (var scope = new TransactionScope())
{
 using (var conn1 = new SqlConnection(connStrDb1))
 {
 conn1.Open();
 SqlCommand cmd1 = conn1.CreateCommand();
 cmd1.CommandText = string.Format("insert into T1 values(1)");
 cmd1.ExecuteNonQuery();
 }

 using (var conn2 = new SqlConnection(connStrDb2))
 {
 conn2.Open();
 var cmd2 = conn2.CreateCommand();
 cmd2.CommandText = string.Format("insert into T2 values(2)");
 cmd2.ExecuteNonQuery();
 }

 scope.Complete();
}
```

## Sharded database applications

Elastic database transactions for SQL DB also support coordinating distributed transactions where you use the `OpenConnectionForKey` method of the elastic database client library to open connections for a scaled out data tier. Consider cases where you need to guarantee transactional consistency for changes across several different sharding key values. Connections to the shards hosting the different sharding key values are brokered using `OpenConnectionForKey`. In the general case, the connections can be to different shards such that ensuring transactional guarantees requires a distributed transaction. The following code sample illustrates this approach. It assumes that a variable called `shardmap` is used to represent a shard map from the elastic database client library:

```
using (var scope = new TransactionScope())
{
 using (var conn1 = shardmap.OpenConnectionForKey(tenantId1, credentialsStr))
 {
 conn1.Open();
 SqlCommand cmd1 = conn1.CreateCommand();
 cmd1.CommandText = string.Format("insert into T1 values(1)");
 cmd1.ExecuteNonQuery();
 }

 using (var conn2 = shardmap.OpenConnectionForKey(tenantId2, credentialsStr))
 {
 conn2.Open();
 var cmd2 = conn2.CreateCommand();
 cmd2.CommandText = string.Format("insert into T1 values(2)");
 cmd2.ExecuteNonQuery();
 }

 scope.Complete();
}
```

## .NET installation for Azure Cloud Services

Azure provides several offerings to host .NET applications. A comparison of the different offerings is available in [Azure App Service, Cloud Services, and Virtual Machines comparison](#). If the guest OS of the offering is smaller than .NET 4.6.1 required for elastic transactions, you need to upgrade the guest OS to 4.6.1.

For Azure App Services, upgrades to the guest OS are currently not supported. For Azure Virtual Machines, simply log into the VM and run the installer for the latest .NET framework. For Azure Cloud Services, you need to include the installation of a newer .NET version into the startup tasks of your deployment. The concepts and steps are documented in [Install .NET on a Cloud Service Role](#).

Note that the installer for .NET 4.6.1 may require more temporary storage during the bootstrapping process on Azure cloud services than the installer for .NET 4.6. To ensure a successful installation, you need to increase temporary storage for your Azure cloud service in your `ServiceDefinition.csdef` file in the `LocalResources` section and the environment settings of your startup task, as shown in the following sample:

```

<LocalResources>
...
 <LocalStorage name="TEMP" sizeInMB="5000" cleanOnRoleRecycle="false" />
 <LocalStorage name="TMP" sizeInMB="5000" cleanOnRoleRecycle="false" />
</LocalResources>
<Startup>
 <Task commandLine="install.cmd" executionContext="elevated" taskType="simple">
 <Environment>
 ...
 <Variable name="TEMP">
 <RoleInstanceValue
 xpath="/RoleEnvironment/CurrentInstance/LocalResources/LocalResource[@name='TEMP']/@path" />
 </Variable>
 <Variable name="TMP">
 <RoleInstanceValue
 xpath="/RoleEnvironment/CurrentInstance/LocalResources/LocalResource[@name='TMP']/@path" />
 </Variable>
 </Environment>
 </Task>
</Startup>

```

## Transactions across multiple servers

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

Elastic database transactions are supported across different SQL Database servers in Azure SQL Database. When transactions cross SQL Database server boundaries, the participating servers first need to be entered into a mutual communication relationship. Once the communication relationship has been established, any database in any of the two servers can participate in elastic transactions with databases from the other server. With transactions spanning more than two SQL Database servers, a communication relationship needs to be in place for any pair of SQL Database servers.

Use the following PowerShell cmdlets to manage cross-server communication relationships for elastic database transactions:

- **New-AzSqlServerCommunicationLink**: Use this cmdlet to create a new communication relationship between two SQL Database servers in Azure SQL Database. The relationship is symmetric which means both servers can initiate transactions with the other server.
- **Get-AzSqlServerCommunicationLink**: Use this cmdlet to retrieve existing communication relationships and their properties.
- **Remove-AzSqlServerCommunicationLink**: Use this cmdlet to remove an existing communication relationship.

## Monitoring transaction status

Use Dynamic Management Views (DMVs) in SQL DB to monitor status and progress of your ongoing elastic database transactions. All DMVs related to transactions are relevant for distributed transactions in SQL DB. You can find the corresponding list of DMVs here: [Transaction Related Dynamic Management Views and Functions \(Transact-SQL\)](#).

These DMVs are particularly useful:

- **sys.dm\_tran\_active\_transactions**: Lists currently active transactions and their status. The UOW (Unit Of Work) column can identify the different child transactions that belong to the same distributed transaction. All transactions within the same distributed transaction carry the same UOW value. See the [DMV documentation](#) for more information.
- **sys.dm\_tran\_database\_transactions**: Provides additional information about transactions, such as placement of the transaction in the log. See the [DMV documentation](#) for more information.
- **sys.dm\_tran\_locks**: Provides information about the locks that are currently held by ongoing transactions. See the [DMV documentation](#) for more information.

## Limitations

The following limitations currently apply to elastic database transactions in SQL DB:

- Only transactions across databases in SQL DB are supported. Other [X/Open XA](#) resource providers and databases outside of SQL DB cannot participate in elastic database transactions. That means that elastic database transactions cannot stretch across on premises SQL Server and Azure SQL Database. For distributed transactions on premises, continue to use MSDTC.
- Only client-coordinated transactions from a .NET application are supported. Server-side support for T-SQL such as BEGIN DISTRIBUTED TRANSACTION is planned, but not yet available.
- Transactions across WCF services are not supported. For example, you have a WCF service method that executes a transaction. Enclosing the call within a transaction scope will fail as a [System.ServiceModel.ProtocolException](#).

## Next steps

For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Multi-model capabilities of Azure SQL Database

11/7/2019 • 8 minutes to read • [Edit Online](#)

Multi-model databases enable you to store and work with data represented in multiple data formats such as relational data, graphs, JSON/XML documents, key-value pairs, etc.

## When to use multi-model capabilities

Azure SQL Database is designed to work with the relational model that provides the best performance in the most of the cases for a variety of general-purpose applications. However, Azure SQL Database is not limited to relational-data only. Azure SQL Database enables you to use a variety of non-relational formats that are tightly integrated into the relational model. You should consider using multi-model capabilities of Azure SQL Database in the following cases:

- You have some information or structures that are better fit for NoSQL models and you don't want to use separate NoSQL database.
- A majority of your data is suitable for relational model, and you need to model some parts of your data in NoSQL style.
- You want to leverage rich Transact-SQL language to query and analyze both relational and NoSQL data, and integrate it with a variety of tools and applications that can use SQL language.
- You want to apply database features such as [in-memory technologies](#) to improve performance of your analytic or processing of your NoSQL data structures, use [transactional replication](#) or [readable replicas](#) to create copy of your data on the other place and offload some analytic workloads from the primary database.

## Overview

Azure SQL provides the following multi-model features:

- [Graph features](#) enable you to represent your data as set of nodes and edges, and use standard Transact-SQL queries enhanced with graph `MATCH` operator to query the graph data.
- [JSON features](#) enable you to put JSON documents in tables, transform relational data to JSON documents and vice versa. You can use the standard Transact-SQL language enhanced with JSON functions for parsing documents, and use non clustered indexes, columnstore indexes, or memory-optimized tables, to optimize your queries.
- [Spatial features](#) enables you to store geographical and geometrical data, index them using the spatial indexes, and retrieve the data using spatial queries.
- [XML features](#) enable you to store and index XML data in your database and use native XQuery/XPath operations to work with XML data. Azure SQL database has specialized built-in XML query engine that process XML data.
- [Key-value pairs](#) are not explicitly supported as special features since key-value pairs can be natively modeled as two-column tables.

#### **NOTE**

You can use JSON Path expression, XQuery/XPath expressions, spatial functions, and graph-query expressions in the same Transact-SQL query to access any data that you stored in the database. Also, any tool or programming language that can execute Transact-SQL queries, can also use that query interface to access multi-model data. This is the key difference compared to the multi-model databases such as [Azure Cosmos DB](#) that provides specialized API for different data models.

In the following sections, you can learn about the most important multi-model capabilities of Azures SQL Database.

## Graph features

Azure SQL Database offers graph database capabilities to model many-to-many relationships in database. A graph is a collection of nodes (or vertices) and edges (or relationships). A node represents an entity (for example, a person or an organization) and an edge represents a relationship between the two nodes that it connects (for example, likes or friends). Here are some features that make a graph database unique:

- Edges or relationships are first class entities in a Graph Database and can have attributes or properties associated with them.
- A single edge can flexibly connect multiple nodes in a Graph Database.
- You can express pattern matching and multi-hop navigation queries easily.
- You can express transitive closure and polymorphic queries easily.

The graph relationships and graph query capabilities are integrated into Transact-SQL and receive the benefits of using SQL Server as the foundational database management system. [Graph processing](#) is the core SQL Server Database Engine feature, so you can find more info about the Graph processing there.

### **When to use a graph capability**

There is nothing a graph database can achieve, which cannot be achieved using a relational database. However, a graph database can make it easier to express certain queries. Your decision to choose one over the other can be based on following factors:

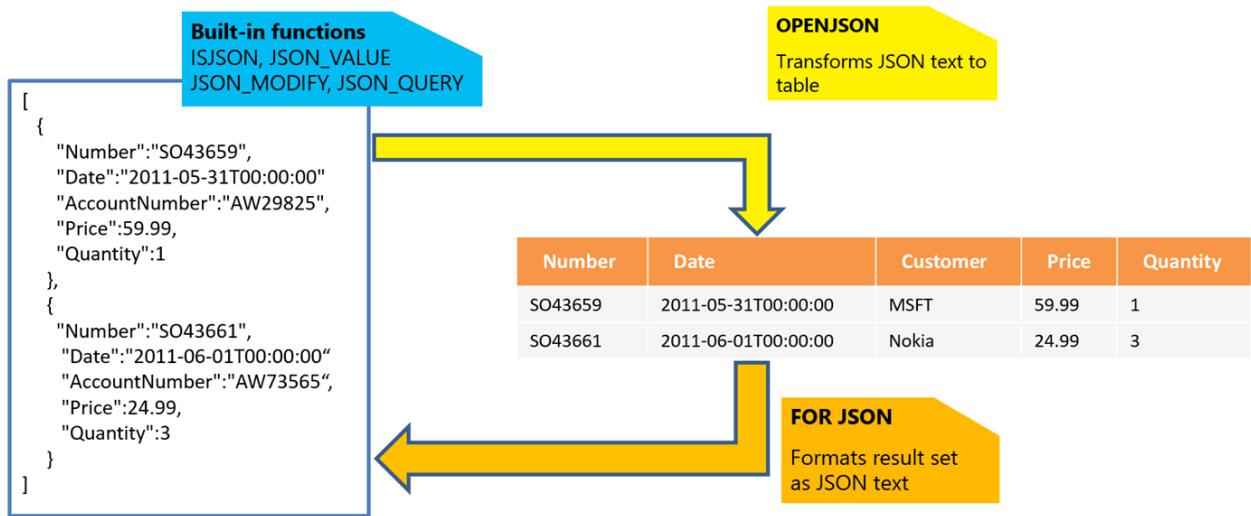
- Model hierarchical data where one node can have multiple parents, so HierarchyId cannot be used
- Model has Your application has complex many-to-many relationships; as application evolves, new relationships are added.
- You need to analyze interconnected data and relationships.

## JSON features

Azure SQL Database lets you parse and query data represented in JavaScript Object Notation ([JSON](#)) format, and export your relational data as JSON text.

JSON is a popular data format used for exchanging data in modern web and mobile applications. JSON is also used for storing semi-structured data in log files or in NoSQL databases like [Azure Cosmos DB](#). Many REST web services return results formatted as JSON text or accept data formatted as JSON. Most Azure services such as [Azure Cognitive Search](#), [Azure Storage](#), and [Azure Cosmos DB](#) have REST endpoints that return or consume JSON.

Azure SQL Database lets you work with JSON data easily and integrate your database with modern services. Azure SQL Database provides the following functions for working with JSON data:



If you have JSON text, you can extract data from JSON or verify that JSON is properly formatted by using the built-in functions `JSON_VALUE`, `JSON_QUERY`, and `ISJSON`. The `JSON_MODIFY` function lets you update value inside JSON text. For more advanced querying and analysis, `OPENJSON` function can transform an array of JSON objects into a set of rows. Any SQL query can be executed on the returned result set. Finally, there is a `FOR JSON` clause that lets you format data stored in your relational tables as JSON text.

For more information, see [How to work with JSON data in azure SQL Database](#). JSON is core SQL Server Database Engine feature, so you can find more info about the JSON feature there.

### When to use a JSON capability

Document models can be used instead of the relational models in some specific scenarios:

- High-normalization of schema doesn't bring significant benefits because you access the all fields of objects at once, or you never update normalized parts of the objects. However, the normalized model increases the complexity of your queries due to the large number of tables that you need to join to get the data.
- You are working with the applications that natively use JSON documents are communication or data models, and you don't want to introduce additional layers that transforms relational data to JSON and vice versa.
- You need to simplify your data model by de-normalizing child tables or Entity-Object-Value patterns.
- You need to load or export data stored in JSON format without some additional tool that parses the data.

## Spatial features

Spatial data represents information about the physical location and shape of geometric objects. These objects can be point locations or more complex objects such as countries/regions, roads, or lakes.

Azure SQL Database supports two spatial data types - the geometry data type and the geography data type.

- The geometry type represents data in a Euclidean (flat) coordinate system.
- The geography type represents data in a round-earth coordinate system.

There is a number of Spatial objects that can be used in Azure SQL database such as `Point`, `LineString`, `Polygon`, etc.

Azure SQL Database also provides specialized `Spatial indexes` that can be used to improve performance of your spatial queries.

`Spatial support` is core SQL Server Database Engine feature, so you can find more info about the spatial feature there.

## XML features

SQL Server provides a powerful platform for developing rich applications for semi-structured data management. Support for XML is integrated into all the components in SQL Server and includes the following:

- The xml data type. XML values can be stored natively in an xml data type column that can be typed according to a collection of XML schemas, or left untyped. You can index the XML column.
- The ability to specify an XQuery query against XML data stored in columns and variables of the xml type. XQuery functionalities can be used in any Transact-SQL query that access any data model that you use in your database.
- Automatically index all elements in XML documents using [primary XML index](#) or specify the exact paths that should be indexed using [secondary XML index](#).
- OPENROWSET that allows bulk loading of XML data.
- Transform relational data to XML format.

[XML](#) is core SQL Server Database Engine feature, so you can find more info about the XML feature there.

### When to use an XML capability

Document models can be used instead of the relational models in some specific scenarios:

- High-normalization of schema doesn't bring significant benefits because you access the all fields of objects at once, or you never update normalized parts of the objects. However, the normalized model increases the complexity of your queries due to the large number of tables that you need to join to get the data.
- You are working with the applications that natively use XML documents are communication or data models, and you don't want to introduce additional layers that transforms relational data to XML and vice versa.
- You need to simplify your data model by de-normalizing child tables or Entity-Object-Value patterns.
- You need to load or export data stored in XML format without some additional tool that parses the data.

## Key-value pairs

Azure SQL Database don't have specialized types or structures that support key-value pairs since key-value structures can be natively represented as standard relational tables:

```
CREATE TABLE Collection (
 Id int identity primary key,
 Data nvarchar(max)
)
```

You can customize this key-value structure to fit your needs without any constraints. As an example, the value can be XML document instead of `nvarchar(max)` type, if the value is JSON document, you can put `CHECK` constraint that verifies the validity of JSON content. You can put any number of values related to one key in the additional columns, add computed columns and indexes to simplify and optimize data access, define the table as memory/optimized schema-only table to get better performance, etc.

See [how BWin is using In-Memory OLTP to achieve unprecedented performance and scale](#) for their ASP.NET caching solution that achieved 1.200.000 batches per seconds, as an example how relational model can be effectively used as key-value pair solution in practice.

## Next steps

Multi-model capabilities in Azure SQL Databases are also the core SQL Server Database Engine features that are shared between Azure SQL Database and SQL Server. To learn more details about these features, visit the SQL Relational database documentation pages:

- [Graph processing](#)
- [JSON data](#)

- Spatial support
- XML data

# Optimize performance by using In-Memory technologies in SQL Database

11/7/2019 • 12 minutes to read • [Edit Online](#)

In-Memory technologies in Azure SQL Database enable you to improve performance of your application, and potentially reduce cost of your database.

## When to use In-Memory technologies

By using In-Memory technologies in Azure SQL Database, you can achieve performance improvements with various workloads:

- **Transactional** (online transactional processing (OLTP)) where most of the requests read or update smaller set of data (for example, CRUD operations).
- **Analytic** (online analytical processing (OLAP)) where most of the queries have complex calculations for the reporting purposes, with a certain number of queries that load and append data to the existing tables (so called bulk-load), or delete the data from the tables.
- **Mixed** (hybrid transaction/analytical processing (HTAP)) where both OLTP and OLAP queries are executed on the same set of data.

In-memory technologies can improve performance of these workloads by keeping the data that should be processed into the memory, using native compilation of the queries, or advanced processing such as batch processing and SIMD instructions that are available on the underlying hardware.

## Overview

Azure SQL Database has the following In-Memory technologies:

- *In-Memory OLTP* increases number of transactions per second and reduces latency for transaction processing. Scenarios that benefit from In-Memory OLTP are: high-throughput transaction processing such as trading and gaming, data ingestion from events or IoT devices, caching, data load, and temporary table and table variable scenarios.
- *Clustered columnstore indexes* reduce your storage footprint (up to 10 times) and improve performance for reporting and analytics queries. You can use it with fact tables in your data marts to fit more data in your database and improve performance. Also, you can use it with historical data in your operational database to archive and be able to query up to 10 times more data.
- *Nonclustered columnstore indexes* for HTAP help you to gain real-time insights into your business through querying the operational database directly, without the need to run an expensive extract, transform, and load (ETL) process and wait for the data warehouse to be populated. Nonclustered columnstore indexes allow fast execution of analytics queries on the OLTP database, while reducing the impact on the operational workload.
- *Memory-optimized clustered columnstore indexes* for HTAP enables you to perform fast transaction processing, and to *concurrently* run analytics queries very quickly on the same data.

Both columnstore indexes and In-Memory OLTP have been part of the SQL Server product since 2012 and 2014, respectively. Azure SQL Database and SQL Server share the same implementation of In-Memory technologies. Going forward, new capabilities for these technologies are released in Azure SQL Database first, before they are released in SQL Server.

## Benefits of In-memory technology

Because of the more efficient query and transaction processing, In-Memory technologies also help you to reduce cost. You typically don't need to upgrade the pricing tier of the database to achieve performance gains. In some cases, you might even be able to reduce the pricing tier, while still seeing performance improvements with In-Memory technologies.

Here are two examples of how In-Memory OLTP helped to significantly improve performance:

- By using In-Memory OLTP, [Quorum Business Solutions was able to double their workload while improving DTUs by 70%](#).
  - DTU means *database transaction unit*, and it includes a measurement of resource consumption.
- The following video demonstrates significant improvement in resource consumption with a sample workload: [In-Memory OLTP in Azure SQL Database Video](#).
  - For more information, see the blog post: [In-Memory OLTP in Azure SQL Database Blog Post](#)

#### NOTE

In-Memory technologies are available in Premium and Business Critical tier Azure SQL databases and Premium elastic pools.

The following video explains potential performance gains with In-Memory technologies in Azure SQL Database. Remember that the performance gain that you see always depends on many factors, including the nature of the workload and data, access pattern of the database, and so on.

This article describes aspects of In-Memory OLTP and columnstore indexes that are specific to Azure SQL Database and also includes samples:

- You'll see the impact of these technologies on storage and data size limits.
- You'll see how to manage the movement of databases that use these technologies between the different pricing tiers.
- You'll see two samples that illustrate the use of In-Memory OLTP, as well as columnstore indexes in Azure SQL Database.

For more information, see:

- [In-Memory OLTP Overview and Usage Scenarios](#) (includes references to customer case studies and information to get started)
- [Documentation for In-Memory OLTP](#)
- [Columnstore Indexes Guide](#)
- Hybrid transactional/analytical processing (HTAP), also known as [real-time operational analytics](#)

## In-memory OLTP

In-memory OLTP technology provides extremely fast data access operations by keeping all data in memory. It also uses specialized indexes, native compilation of queries, and latch-free data-access to improve performance of the OLTP workload. There are two ways to organize your In-Memory OLTP data:

- **Memory-optimized rowstore** format where every row is a separate memory object. This is a classic In-Memory OLTP format optimized for high-performance OLTP workloads. There are two types of memory-optimized tables that can be used in the memory-optimized rowstore format:
  - *Durable tables* (SCHEMA\_AND\_DATA) where the rows placed in memory are preserved after server restart. This type of tables behaves like a traditional rowstore table with the additional benefits of in-

memory optimizations.

- *Non-durable tables* (SCHEMA\_ONLY) where the rows are not-preserved after restart. This type of table is designed for temporary data (for example, replacement of temp tables), or tables where you need to quickly load data before you move it to some persisted table (so called staging tables).
- **Memory-optimized columnstore** format where data is organized in a columnar format. This structure is designed for HTAP scenarios where you need to run analytic queries on the same data structure where your OLTP workload is running.

#### NOTE

In-Memory OLTP technology is designed for the data structures that can fully reside in memory. Since the In-memory data cannot be offloaded to disk, make sure that you are using database that has enough memory. See [Data size and storage cap for In-Memory OLTP](#) for more details.

A quick primer on In-Memory OLTP: [Quickstart 1: In-Memory OLTP Technologies for Faster T-SQL Performance](#) (another article to help you get started)

In-depth videos about the technologies:

- [In-Memory OLTP in Azure SQL Database](#) (which contains a demo of performance benefits and steps to reproduce these results yourself)
- [In-Memory OLTP Videos: What it is and When/How to use it](#)

There is a programmatic way to understand whether a given database supports In-Memory OLTP. You can execute the following Transact-SQL query:

```
SELECT DatabasePropertyEx(DB_NAME(), 'IsXTPSupported');
```

If the query returns **1**, In-Memory OLTP is supported in this database. The following queries identify all objects that need to be removed before a database can be downgraded to Standard/Basic:

```
SELECT * FROM sys.tables WHERE is_memory_optimized=1
SELECT * FROM sys.table_types WHERE is_memory_optimized=1
SELECT * FROM sys.sql_modules WHERE uses_native_compilation=1
```

#### Data size and storage cap for In-Memory OLTP

In-Memory OLTP includes memory-optimized tables, which are used for storing user data. These tables are required to fit in memory. Because you manage memory directly in the SQL Database service, we have the concept of a quota for user data. This idea is referred to as *In-Memory OLTP storage*.

Each supported single database pricing tier and each elastic pool pricing tier includes a certain amount of In-Memory OLTP storage. See [DTU-based resource limits - single database](#), [DTU-based resource limits - elastic pools](#), [vCore-based resource limits - single databases](#) and [vCore-based resource limits - elastic pools](#).

The following items count toward your In-Memory OLTP storage cap:

- Active user data rows in memory-optimized tables and table variables. Note that old row versions don't count toward the cap.
- Indexes on memory-optimized tables.
- Operational overhead of ALTER TABLE operations.

If you hit the cap, you receive an out-of-quota error, and you are no longer able to insert or update data. To mitigate this error, delete data or increase the pricing tier of the database or pool.

For details about monitoring In-Memory OLTP storage utilization and configuring alerts when you almost hit the cap, see [Monitor In-Memory storage](#).

#### About elastic pools

With elastic pools, the In-Memory OLTP storage is shared across all databases in the pool. Therefore, the usage in one database can potentially affect other databases. Two mitigations for this are:

- Configure a `Max-eDTU` or `MaxvCore` for databases that is lower than the eDTU or vCore count for the pool as a whole. This maximum caps the In-Memory OLTP storage utilization, in any database in the pool, to the size that corresponds to the eDTU count.
- Configure a `Min-eDTU` or `MinvCore` that is greater than 0. This minimum guarantees that each database in the pool has the amount of available In-Memory OLTP storage that corresponds to the configured `Min-eDTU` or `vCore`.

#### Changing service tiers of databases that use In-Memory OLTP technologies

You can always upgrade your database or instance to a higher tier, such as from General Purpose to Business Critical (or Standard to Premium). The available functionality and resources only increase.

But downgrading the tier can negatively impact your database. The impact is especially apparent when you downgrade from Business Critical to General Purpose (or Premium to Standard or Basic) when your database contains In-Memory OLTP objects. Memory-optimized tables are unavailable after the downgrade (even if they remain visible). The same considerations apply when you're lowering the pricing tier of an elastic pool, or moving a database with In-Memory technologies, into a Standard or Basic elastic pool.

##### IMPORTANT

In-Memory OLTP isn't supported in the General Purpose, Standard or Basic tier. Therefore, it isn't possible to move a database that has any In-Memory OLTP objects to the Standard or Basic tier.

Before you downgrade the database to Standard/Basic, remove all memory-optimized tables and table types, as well as all natively compiled T-SQL modules.

*Scaling-down resources in Business Critical tier:* Data in memory-optimized tables must fit within the In-Memory OLTP storage that is associated with the tier of the database or Managed Instance, or it is available in the elastic pool. If you try to scale-down the tier or move the database into a pool that doesn't have enough available In-Memory OLTP storage, the operation fails.

## In-memory columnstore

In-memory columnstore technology is enabling you to store and query a large amount of data in the tables. Columnstore technology uses column-based data storage format and batch query processing to achieve gain up to 10 times the query performance in OLAP workloads over traditional row-oriented storage. You can also achieve gains up to 10 times the data compression over the uncompressed data size. There are two types of columnstore models that you can use to organize your data:

- Clustered columnstore** where all data in the table is organized in the columnar format. In this model, all rows in the table are placed in columnar format that highly compresses the data and enables you to execute fast analytical queries and reports on the table. Depending on the nature of your data, the size of your data might be decreased 10x-100x. Clustered columnstore model also enables fast ingestion of large amount of data (bulk-load) since large batches of data greater than 100K rows are compressed before they are stored on disk. This model is a good choice for the classic data warehouse scenarios.
- Non-clustered columnstore** where the data is stored in traditional rowstore table and there is an index in the columnstore format that is used for the analytical queries. This model enables Hybrid Transactional-Analytic Processing (HTAP): the ability to run performant real-time analytics on a transactional workload.

OLTP queries are executed on rowstore table that is optimized for accessing a small set of rows, while OLAP queries are executed on columnstore index that is better choice for scans and analytics. Azure SQL Database Query optimizer dynamically chooses rowstore or columnstore format based on the query. Non-clustered columnstore indexes don't decrease the size of the data since original data-set is kept in the original rowstore table without any change. However, the size of additional columnstore index should be in order of magnitude smaller than the equivalent B-tree index.

#### NOTE

In-memory columnstore technology keeps only the data that is needed for processing in the memory, while the data that cannot fit into the memory is stored on-disk. Therefore, the amount of data in In-memory columnstore structures can exceed the amount of available memory.

In-depth video about the technology:

- [Columnstore Index: In-Memory Analytics Videos from Ignite 2016](#)

#### Data size and storage for columnstore indexes

Columnstore indexes aren't required to fit in memory. Therefore, the only cap on the size of the indexes is the maximum overall database size, which is documented in the [DTU-based purchasing model](#) and [vCore-based purchasing model](#) articles.

When you use clustered columnstore indexes, columnar compression is used for the base table storage. This compression can significantly reduce the storage footprint of your user data, which means that you can fit more data in the database. And the compression can be further increased with [columnar archival compression](#). The amount of compression that you can achieve depends on the nature of the data, but 10 times the compression is not uncommon.

For example, if you have a database with a maximum size of 1 terabyte (TB) and you achieve 10 times the compression by using columnstore indexes, you can fit a total of 10 TB of user data in the database.

When you use nonclustered columnstore indexes, the base table is still stored in the traditional rowstore format. Therefore, the storage savings aren't as big as with clustered columnstore indexes. However, if you're replacing a number of traditional nonclustered indexes with a single columnstore index, you can still see an overall savings in the storage footprint for the table.

#### Changing service tiers of databases containing Columnstore indexes

*Downgrading single database to Basic or Standard* might not be possible if your target tier is below S3. Columnstore indexes are supported only on the Business Critical/Premium pricing tier and on the Standard tier, S3 and above, and not on the Basic tier. When you downgrade your database to an unsupported tier or level, your columnstore index becomes unavailable. The system maintains your columnstore index, but it never leverages the index. If you later upgrade back to a supported tier or level, your columnstore index is immediately ready to be leveraged again.

If you have a **clustered** columnstore index, the whole table becomes unavailable after the downgrade. Therefore we recommend that you drop all *clustered* columnstore indexes before you downgrade your database to an unsupported tier or level.

#### NOTE

Managed Instance supports ColumnStore indexes in all tiers.

## Next steps

- [Quickstart 1: In-Memory OLTP Technologies for faster T-SQL Performance](#)
- [Use In-Memory OLTP in an existing Azure SQL application](#)
- [Monitor In-Memory OLTP storage for In-Memory OLTP](#)
- [Try In-memory features in Azure SQL Database](#)

## Additional resources

### Deeper information

- [Learn how Quorum doubles key database's workload while lowering DTU by 70% with In-Memory OLTP in SQL Database](#)
- [In-Memory OLTP in Azure SQL Database Blog Post](#)
- [Learn about In-Memory OLTP](#)
- [Learn about columnstore indexes](#)
- [Learn about real-time operational analytics](#)
- See [Common Workload Patterns and Migration Considerations](#) (which describes workload patterns where In-Memory OLTP commonly provides significant performance gains)

### Application design

- [In-Memory OLTP \(In-Memory Optimization\)](#)
- [Use In-Memory OLTP in an existing Azure SQL application](#)

### Tools

- [Azure portal](#)
- [SQL Server Management Studio \(SSMS\)](#)
- [SQL Server Data Tools \(SSDT\)](#)

# Use In-Memory OLTP to improve your application performance in SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

In-Memory OLTP can be used to improve the performance of transaction processing, data ingestion, and transient data scenarios, in **Premium and Business Critical tier** databases without increasing the pricing tier.

## NOTE

Learn how [Quorum doubles key database's workload while lowering DTU by 70% with SQL Database](#)

Follow these steps to adopt In-Memory OLTP in your existing database.

## Step 1: Ensure you are using a Premium and Business Critical tier database

In-Memory OLTP is supported only in Premium and Business Critical tier databases. In-Memory is supported if the returned result is 1 (not 0):

```
SELECT DatabasePropertyEx(Db_Name(), 'IsXTPSupported');
```

XTP stands for *Extreme Transaction Processing*

## Step 2: Identify objects to migrate to In-Memory OLTP

SSMS includes a **Transaction Performance Analysis Overview** report that you can run against a database with an active workload. The report identifies tables and stored procedures that are candidates for migration to In-Memory OLTP.

In SSMS, to generate the report:

- In the **Object Explorer**, right-click your database node.
- Click **Reports > Standard Reports > Transaction Performance Analysis Overview**.

For more information, see [Determining if a Table or Stored Procedure Should Be Ported to In-Memory OLTP](#).

## Step 3: Create a comparable test database

Suppose the report indicates your database has a table that would benefit from being converted to a memory-optimized table. We recommend that you first test to confirm the indication by testing.

You need a test copy of your production database. The test database should be at the same service tier level as your production database.

To ease testing, tweak your test database as follows:

1. Connect to the test database by using SSMS.
2. To avoid needing the WITH (SNAPSHOT) option in queries, set the database option as shown in the following T-SQL statement:

```
ALTER DATABASE CURRENT
SET
 MEMORY_OPTIMIZED_ELEVATE_TO_SNAPSHOT = ON;
```

## Step 4: Migrate tables

You must create and populate a memory-optimized copy of the table you want to test. You can create it by using either:

- The handy Memory Optimization Wizard in SSMS.
- Manual T-SQL.

### Memory Optimization Wizard in SSMS

To use this migration option:

1. Connect to the test database with SSMS.
2. In the **Object Explorer**, right-click on the table, and then click **Memory Optimization Advisor**.
  - The **Table Memory Optimizer Advisor** wizard is displayed.
3. In the wizard, click **Migration validation** (or the **Next** button) to see if the table has any unsupported features that are unsupported in memory-optimized tables. For more information, see:
  - The *memory optimization checklist* in [Memory Optimization Advisor](#).
  - [Transact-SQL Constructs Not Supported by In-Memory OLTP](#).
  - [Migrating to In-Memory OLTP](#).
4. If the table has no unsupported features, the advisor can perform the actual schema and data migration for you.

### Manual T-SQL

To use this migration option:

1. Connect to your test database by using SSMS (or a similar utility).
2. Obtain the complete T-SQL script for your table and its indexes.
  - In SSMS, right-click your table node.
  - Click **Script Table As > CREATE To > New Query Window**.
3. In the script window, add WITH (MEMORY\_OPTIMIZED = ON) to the CREATE TABLE statement.
4. If there is a CLUSTERED index, change it to NONCLUSTERED.
5. Rename the existing table by using SP\_RENAME.
6. Create the new memory-optimized copy of the table by running your edited CREATE TABLE script.
7. Copy the data to your memory-optimized table by using INSERT...SELECT \* INTO:

```
INSERT INTO <new_memory_optimized_table>
SELECT * FROM <old_disk_based_table>;
```

## Step 5 (optional): Migrate stored procedures

The In-Memory feature can also modify a stored procedure for improved performance.

### Considerations with natively compiled stored procedures

A natively compiled stored procedure must have the following options on its T-SQL WITH clause:

- NATIVE\_COMPILATION
- SCHEMABINDING: meaning tables that the stored procedure cannot have their column definitions changed in any way that would affect the stored procedure, unless you drop the stored procedure.

A native module must use one big [ATOMIC blocks](#) for transaction management. There is no role for an explicit BEGIN TRANSACTION, or for ROLLBACK TRANSACTION. If your code detects a violation of a business rule, it can terminate the atomic block with a [THROW](#) statement.

### Typical CREATE PROCEDURE for natively compiled

Typically the T-SQL to create a natively compiled stored procedure is similar to the following template:

```
CREATE PROCEDURE schemaname.procedurename
 @param1 type1, ...
 WITH NATIVE_COMPILATION, SCHEMABINDING
 AS
 BEGIN ATOMIC WITH
 (TRANSACTION ISOLATION LEVEL = SNAPSHOT,
 LANGUAGE = N'your_language__see_sys.languages'
)
 ...
 END;
```

- For the TRANSACTION\_ISOLATION\_LEVEL, SNAPSHOT is the most common value for the natively compiled stored procedure. However, a subset of the other values are also supported:
  - REPEATABLE READ
  - SERIALIZABLE
- The LANGUAGE value must be present in the sys.languages view.

### How to migrate a stored procedure

The migration steps are:

1. Obtain the CREATE PROCEDURE script to the regular interpreted stored procedure.
2. Rewrite its header to match the previous template.
3. Ascertain whether the stored procedure T-SQL code uses any features that are not supported for natively compiled stored procedures. Implement workarounds if necessary.
  - For details see [Migration Issues for Natively Compiled Stored Procedures](#).
4. Rename the old stored procedure by using SP\_RENAME. Or simply DROP it.
5. Run your edited CREATE PROCEDURE T-SQL script.

## Step 6: Run your workload in test

Run a workload in your test database that is similar to the workload that runs in your production database. This should reveal the performance gain achieved by your use of the In-Memory feature for tables and stored procedures.

Major attributes of the workload are:

- Number of concurrent connections.
- Read/write ratio.

To tailor and run the test workload, consider using the handy ostress.exe tool, which illustrated in [here](#).

To minimize network latency, run your test in the same Azure geographic region where the database exists.

## Step 7: Post-implementation monitoring

Consider monitoring the performance effects of your In-Memory implementations in production:

- [Monitor In-Memory Storage](#).
- [Monitoring Azure SQL Database using dynamic management views](#)

## Related links

- [In-Memory OLTP \(In-Memory Optimization\)](#)
- [Introduction to Natively Compiled Stored Procedures](#)
- [Memory Optimization Advisor](#)

# Transactional replication with single, pooled, and instance databases in Azure SQL Database

1/6/2020 • 7 minutes to read • [Edit Online](#)

Transactional replication is a feature of Azure SQL Database and SQL Server that enables you to replicate data from a table in Azure SQL Database or a SQL Server to the tables placed on remote databases. This feature allows you to synchronize multiple tables in different databases.

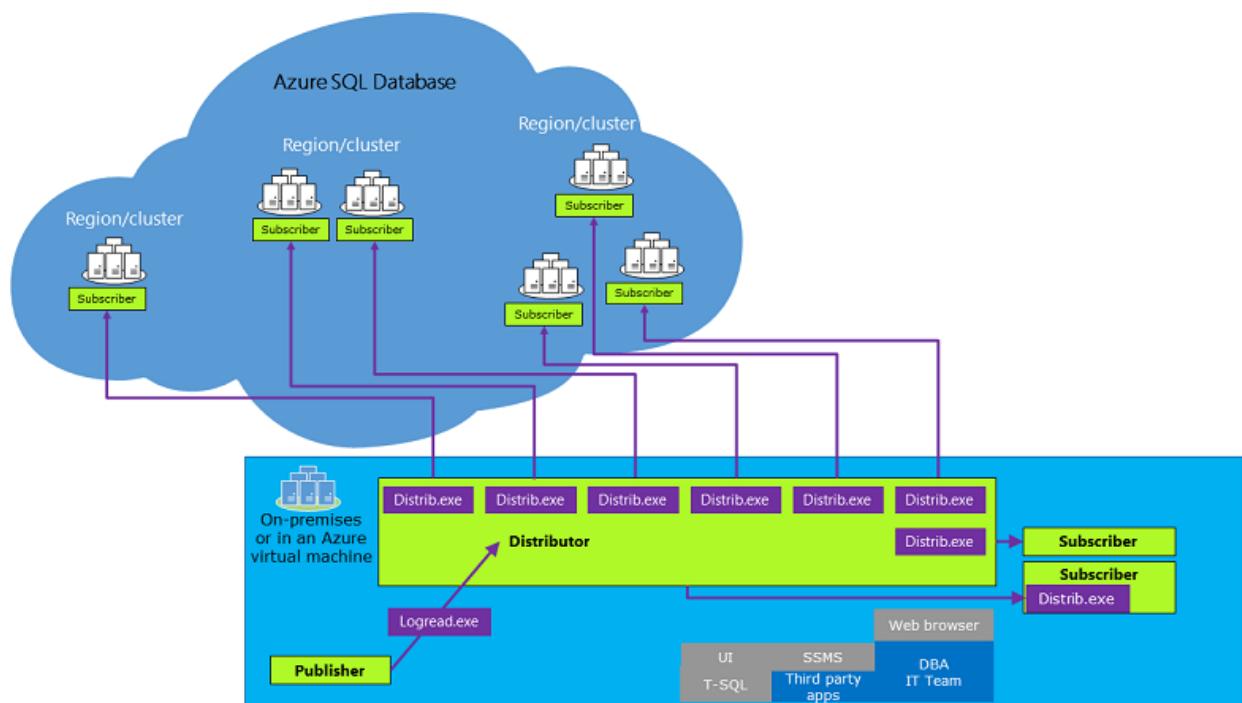
## When to use Transactional replication

Transactional replication is useful in the following scenarios:

- Publish changes made in one or more tables in a database and distribute them to one or many SQL Server or Azure SQL databases that subscribed for the changes.
- Keep several distributed databases in synchronized state.
- Migrate databases from one SQL Server or managed instance to another database by continuously publishing the changes.

## Overview

The key components in transactional replication are shown in the following picture:



The **Publisher** is an instance or server that publishes changes made on some tables (articles) by sending the updates to the Distributor. Publishing to any Azure SQL database from an on-premises SQL Server is supported by the following versions of SQL Server:

- SQL Server 2019 (preview)
- SQL Server 2016 to SQL 2017
- SQL Server 2014 SP1 CU3 or greater (12.00.4427)
- SQL Server 2014 RTM CU10 (12.00.2556)

- SQL Server 2012 SP3 or greater (11.0.6020)
- SQL Server 2012 SP2 CU8 (11.0.5634.0)
- For other versions of SQL Server that do not support publishing to objects in Azure, it is possible to utilize the [republishing data](#) method to move data to newer versions of SQL Server.

The **Distributor** is an instance or server that collects changes in the articles from a Publisher and distributes them to the Subscribers. The Distributor can be either Azure SQL Database managed instance or SQL Server (any version as long it is equal to or higher than the Publisher version).

The **Subscriber** is an instance or server that is receiving the changes made on the Publisher. Subscribers can be either single, pooled, and instance databases in Azure SQL Database or SQL Server databases. A Subscriber on a single or pooled database must be configured as push-subscriber.

ROLE	SINGLE AND POOLED DATABASES	INSTANCE DATABASES
<b>Publisher</b>	No	Yes
<b>Distributor</b>	No	Yes
<b>Pull subscriber</b>	No	Yes
<b>Push Subscriber</b>	Yes	Yes

#### NOTE

A pull subscription is not supported when the distributor is an Instance database and the subscriber is not.

There are different [types of replication](#):

REPLICATION	SINGLE AND POOLED DATABASES	INSTANCE DATABASES
<b>Standard Transactional</b>	Yes (only as subscriber)	Yes
<b>Snapshot</b>	Yes (only as subscriber)	Yes
<b>Merge replication</b>	No	No
<b>Peer-to-peer</b>	No	No
<b>Bidirectional</b>	No	Yes
<b>Updatable subscriptions</b>	No	No

**NOTE**

- Attempting to configure replication using an older version can result in error number MSSQL\_REPL20084 (The process could not connect to Subscriber.) and MSSQ\_REPL40532 (Cannot open server <name> requested by the login. The login failed.)
- To use all the features of Azure SQL Database, you must be using the latest versions of [SQL Server Management Studio \(SSMS\)](#) and [SQL Server Data Tools \(SSDT\)](#).

**Supportability matrix for Instance Databases and On-premises systems**

The replication supportability matrix for instance databases is the same as the one for SQL Server on-premises.

PUBLISHER	DISTRIBUTOR	SUBSCRIBER
SQL Server 2019	SQL Server 2019	SQL Server 2019 SQL Server 2017 SQL Server 2016
SQL Server 2017	SQL Server 2019 SQL Server 2017	SQL Server 2019 SQL Server 2017 SQL Server 2016 SQL Server 2014
SQL Server 2016	SQL Server 2019 SQL Server 2017 SQL Server 2016	SQL Server 2019 SQL Server 2017 SQL Server 2016 SQL Server 2014 SQL Server 2012
SQL Server 2014	SQL Server 2019 SQL Server 2017 SQL Server 2016 SQL Server 2014	SQL Server 2017 SQL Server 2016 SQL Server 2014 SQL Server 2012 SQL Server 2008 R2 SQL Server 2008
SQL Server 2012	SQL Server 2019 SQL Server 2017 SQL Server 2016 SQL Server 2014 SQL Server 2012	SQL Server 2016 SQL Server 2014 SQL Server 2012 SQL Server 2008 R2 SQL Server 2008
SQL Server 2008 R2 SQL Server 2008	SQL Server 2019 SQL Server 2017 SQL Server 2016 SQL Server 2014 SQL Server 2012 SQL Server 2008 R2 SQL Server 2008	SQL Server 2014 SQL Server 2012 SQL Server 2008 R2 SQL Server 2008

## Requirements

- Connectivity uses SQL Authentication between replication participants.
- An Azure Storage Account share for the working directory used by replication.
- Port 445 (TCP outbound) needs to be open in the security rules of the managed instance subnet to access

the Azure file share.

- Port 1433 (TCP outbound) needs to be opened if the Publisher/Distributor are on a managed instance and the subscriber is on-premises.
- All types of replication participants (Publisher, Distributor, Pull Subscriber, and Push Subscriber) can be placed on managed instances, but the publisher and the distributor must be either both in the cloud or both on-premises.
- If either the publisher, distributor, and/or the subscriber exist in different virtual networks, then VPN peering must be established between each entity, such that there is VPN peering between the publisher and distributor, and/or there is VPN peering between the distributor and subscriber.

**NOTE**

- You may encounter error 53 when connecting to an Azure Storage File if the outbound network security group (NSG) port 445 is blocked when the distributor is an instance database and the subscriber is on-premises. [Update the vNet NSG](#) to resolve this issue.

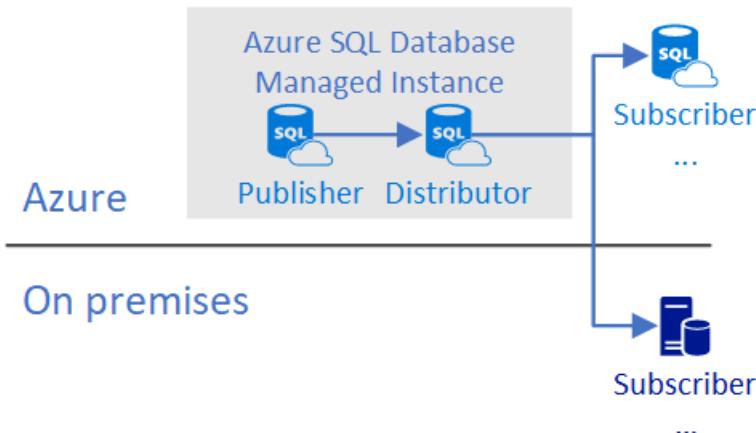
### Compare Data Sync with Transactional Replication

	DATA SYNC	TRANSACTIONAL REPLICATION
Advantages	<ul style="list-style-type: none"><li>- Active-active support</li><li>- Bi-directional between on-premises and Azure SQL Database</li></ul>	<ul style="list-style-type: none"><li>- Lower latency</li><li>- Transactional consistency</li><li>- Reuse existing topology after migration</li></ul>
Disadvantages	<ul style="list-style-type: none"><li>- 5 min or more latency</li><li>- No transactional consistency</li><li>- Higher performance impact</li></ul>	<ul style="list-style-type: none"><li>- Can't publish from Azure SQL Database single database or pooled database</li><li>- High maintenance cost</li></ul>

## Common configurations

In general, the publisher and the distributor must be either in the cloud or on-premises. The following configurations are supported:

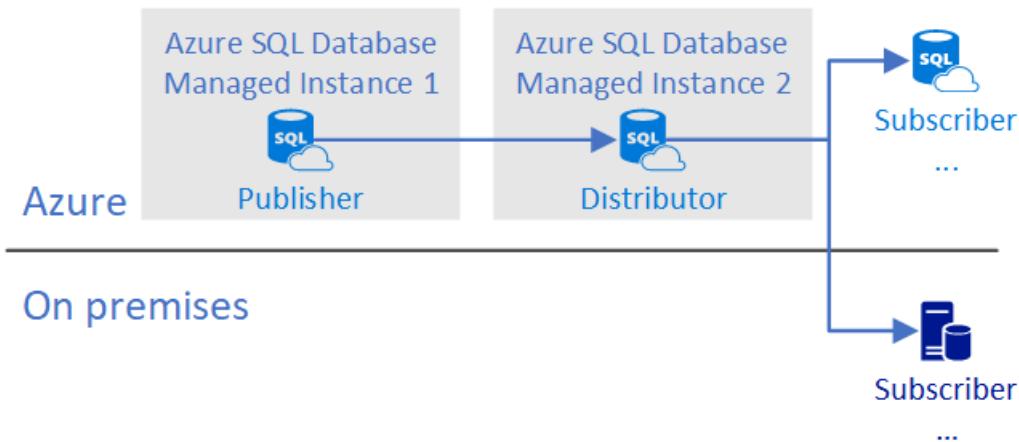
### Publisher with local Distributor on a managed instance



Publisher and distributor are configured within a single managed instance and distributing changes to other managed instance, single database, pooled database, or SQL Server on-premises.

### Publisher with remote distributor on a managed instance

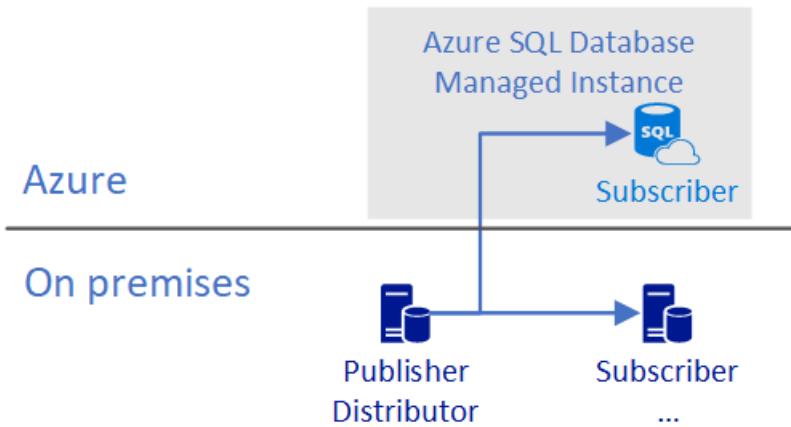
In this configuration, one managed instance publishes changes to distributor placed on another managed instance that can serve many source managed instances and distribute changes to one or many targets on managed instance, single database, pooled database, or SQL Server.



Publisher and distributor are configured on two managed instances. There are some constraints with this configuration:

- Both managed instances are on the same vNet.
- Both managed instances are in the same location.

#### **Publisher and distributor on-premises with a subscriber on a single, pooled, and instance database**



In this configuration, an Azure SQL Database (single, pooled, and instance database) is a subscriber. This configuration supports migration from on-premises to Azure. If a subscriber is on a single or pooled database, it must be in push mode.

## With failover groups

If geo-replication is enabled on a **publisher** or **distributor** instance in a **failover group**, the managed instance administrator must clean up all publications on the old primary and reconfigure them on the new primary after a failover occurs. The following activities are needed in this scenario:

1. Stop all replication jobs running on the database, if there are any.
2. Drop subscription metadata from publisher by running the following script on publisher database:

```
EXEC sp_dropsubscription @publication='<name of publication>', @article='all',@subscriber='<name of subscriber>'
```
3. Drop subscription metadata from the subscriber. Run the following script on the subscription database on subscriber instance:

```
EXEC sp_subscription_cleanup
 @publisher = N'<full DNS of publisher, e.g. example.ac2d23028af5.database.windows.net>',
 @publisher_db = N'<publisher database>',
 @publication = N'<name of publication>';
```

4. Forcefully drop all replication objects from publisher by running the following script in the published database:

```
EXEC sp_removedbreplication
```

5. Forcefully drop old distributor from original primary instance (if failing back over to an old primary that used to have a distributor). Run the following script on the master database in old distributor managed instance:

```
EXEC sp_dropdistributor 1,1
```

If geo-replication is enabled on a **subscriber** instance in a failover group, the publication should be configured to connect to the failover group listener endpoint for the subscriber managed instance. In the event of a failover, subsequent action by the managed instance administrator depends on the type of failover that occurred:

- For a failover with no data loss, replication will continue working after failover.
- For a failover with data loss, replication will work as well. It will replicate the lost changes again.
- For a failover with data loss, but the data loss is outside of the distribution database retention period, the managed instance administrator will need to reinitialize the subscription database.

## Next steps

- [Configure replication between an MI publisher and subscriber](#)
- [Configure replication between an MI publisher, MI distributor, and SQL Server subscriber](#)
- [Create a publication](#).
- [Create a push subscription](#) by using the Azure SQL Database server name as the subscriber (for example `N'azuresqlfdbdns.database.windows.net`) and the Azure SQL Database name as the destination database (for example **Adventureworks**.)

For more information about configuring transactional replication, see the following tutorials:

## See Also

- [Replication with an MI and a failover group](#)
- [Replication to SQL Database](#)
- [Replication to managed instance](#)
- [Create a Publication](#)
- [Create a Push Subscription](#)
- [Types of Replication](#)
- [Monitoring \(Replication\)](#)
- [Initialize a Subscription](#)

# Getting Started with Temporal Tables in Azure SQL Database

11/7/2019 • 7 minutes to read • [Edit Online](#)

Temporal Tables are a new programmability feature of Azure SQL Database that allows you to track and analyze the full history of changes in your data, without the need for custom coding. Temporal Tables keep data closely related to time context so that stored facts can be interpreted as valid only within the specific period. This property of Temporal Tables allows for efficient time-based analysis and getting insights from data evolution.

## Temporal Scenario

This article illustrates the steps to utilize Temporal Tables in an application scenario. Suppose that you want to track user activity on a new website that is being developed from scratch or on an existing website that you want to extend with user activity analytics. In this simplified example, we assume that the number of visited web pages during a period of time is an indicator that needs to be captured and monitored in the website database that is hosted on Azure SQL Database. The goal of the historical analysis of user activity is to get inputs to redesign website and provide better experience for the visitors.

The database model for this scenario is very simple - user activity metric is represented with a single integer field, **PageVisited**, and is captured along with basic information on the user profile. Additionally, for time-based analysis, you would keep a series of rows for each user, where every row represents the number of pages a particular user visited within a specific period of time.



Fortunately, you do not need to put any effort in your app to maintain this activity information. With Temporal Tables, this process is automated - giving you full flexibility during website design and more time to focus on the data analysis itself. The only thing you have to do is to ensure that **WebSiteInfo** table is configured as [temporal system-versioned](#). The exact steps to utilize Temporal Tables in this scenario are described below.

## Step 1: Configure tables as temporal

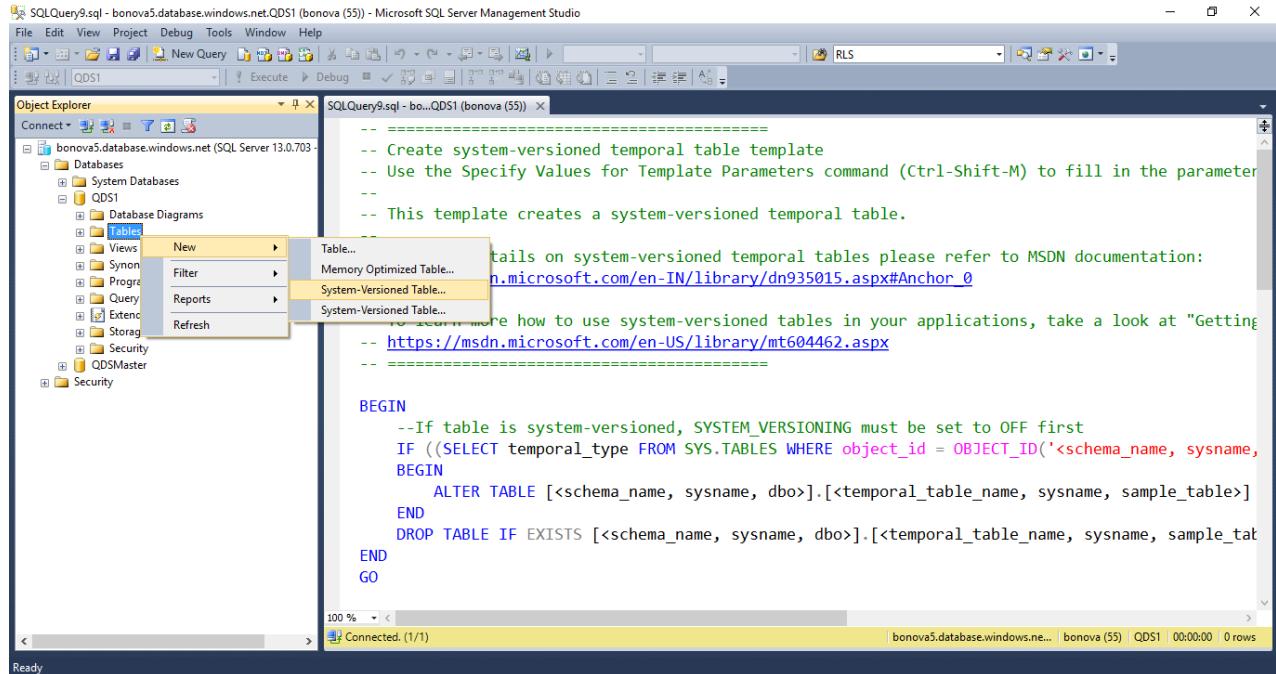
Depending on whether you are starting new development or upgrading existing application, you will either create temporal tables or modify existing ones by adding temporal attributes. In general case, your scenario can be a mix of these two options. Perform these action using [SQL Server Management Studio \(SSMS\)](#), [SQL Server Data Tools \(SSDT\)](#) or any other Transact-SQL development tool.

## IMPORTANT

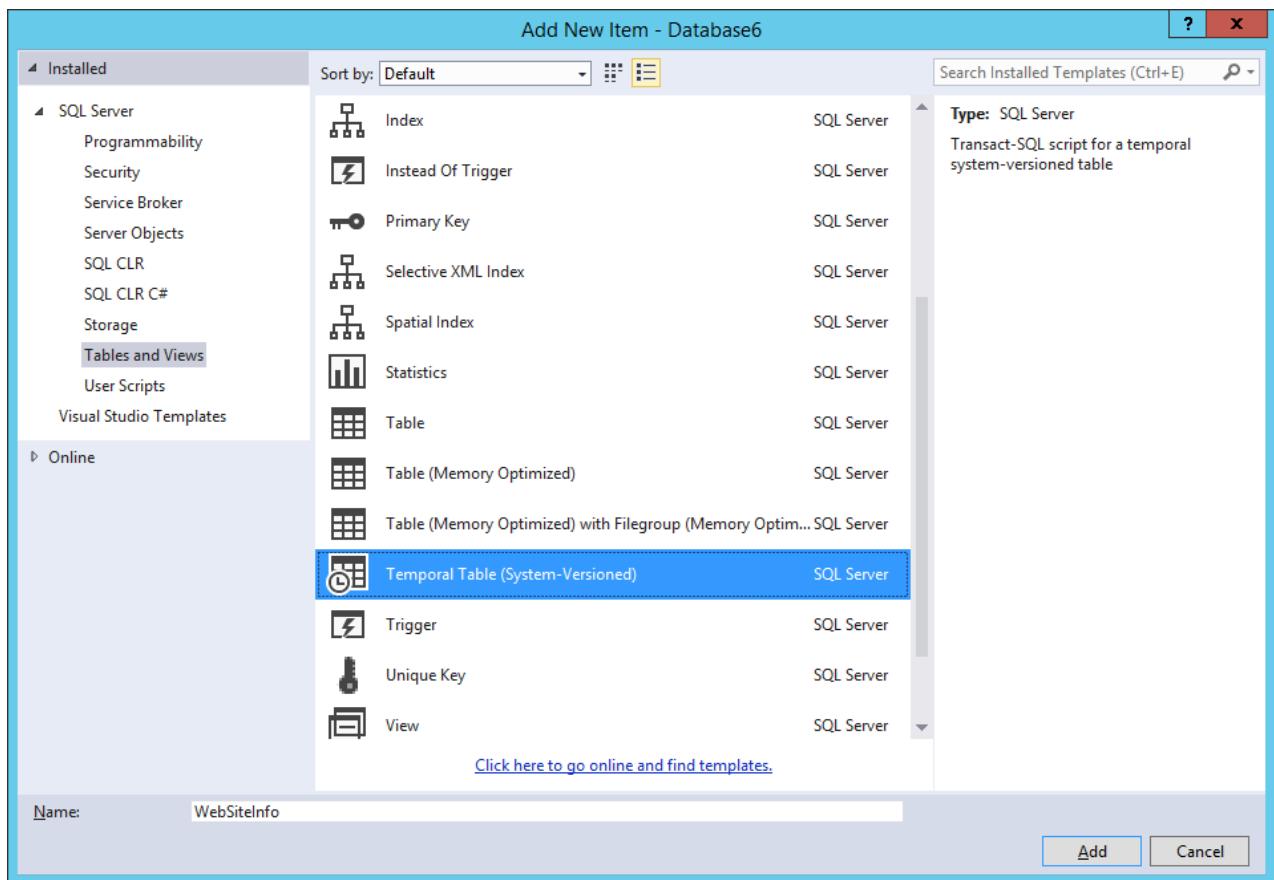
It is recommended that you always use the latest version of Management Studio to remain synchronized with updates to Microsoft Azure and SQL Database. [Update SQL Server Management Studio](#).

## Create new table

Use context menu item "New System-Versioned Table" in SSMS Object Explorer to open the query editor with a temporal table template script and then use "Specify Values for Template Parameters" (Ctrl+Shift+M) to populate the template:



In SSDT, choose "Temporal Table (System-Versioned)" template when adding new items to the database project. That will open table designer and enable you to easily specify the table layout:



You can also create temporal table by specifying the Transact-SQL statements directly, as shown in the example below. Note that the mandatory elements of every temporal table are the PERIOD definition and the SYSTEM\_VERSIONING clause with a reference to another user table that will store historical row versions:

```
CREATE TABLE WebsiteUserInfo
(
 [UserID] int NOT NULL PRIMARY KEY CLUSTERED
 , [UserName] nvarchar(100) NOT NULL
 , [PagesVisited] int NOT NULL
 , [ValidFrom] datetime2 (0) GENERATED ALWAYS AS ROW START
 , [ValidTo] datetime2 (0) GENERATED ALWAYS AS ROW END
 , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.WebsiteUserInfoHistory));
```

When you create system-versioned temporal table, the accompanying history table with the default configuration is automatically created. The default history table contains a clustered B-tree index on the period columns (end, start) with page compression enabled. This configuration is optimal for the majority of scenarios in which temporal tables are used, especially for [data auditing](#).

In this particular case, we aim to perform time-based trend analysis over a longer data history and with bigger data sets, so the storage choice for the history table is a clustered columnstore index. A clustered columnstore provides very good compression and performance for analytical queries. Temporal Tables give you the flexibility to configure indexes on the current and temporal tables completely independently.

#### NOTE

Columnstore indexes are available in the Premium tier and in the Standard tier, S3 and above.

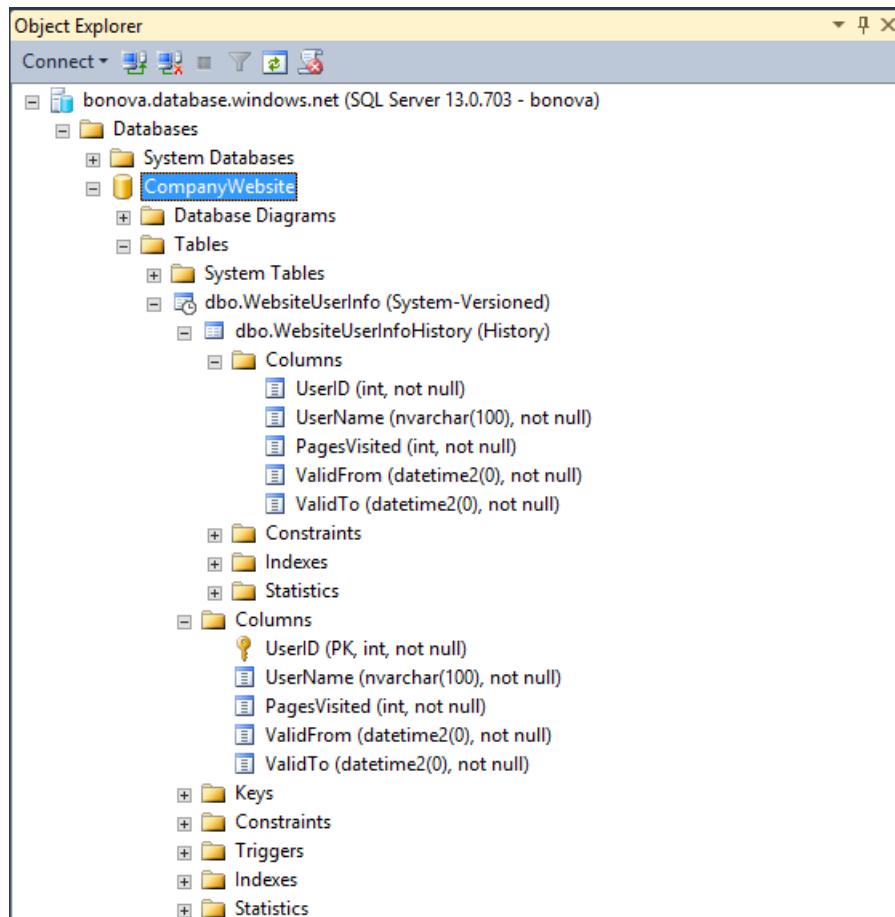
The following script shows how default index on history table can be changed to the clustered columnstore:

```

CREATE CLUSTERED COLUMNSTORE INDEX IX_WebsiteUserInfoHistory
ON dbo.WebsiteUserInfoHistory
WITH (DROP_EXISTING = ON);

```

Temporal Tables are represented in the Object Explorer with the specific icon for easier identification, while its history table is displayed as a child node.



### Alter existing table to temporal

Let's cover the alternative scenario in which the WebsiteUserInfo table already exists, but was not designed to keep a history of changes. In this case, you can simply extend the existing table to become temporal, as shown in the following example:

```

ALTER TABLE WebsiteUserInfo
ADD
 ValidFrom datetime2 (0) GENERATED ALWAYS AS ROW START HIDDEN
 constraint DF_ValidFrom DEFAULT DATEADD(SECOND, -1, SYSUTCDATETIME())
 , ValidTo datetime2 (0) GENERATED ALWAYS AS ROW END HIDDEN
 constraint DF_ValidTo DEFAULT '9999.12.31 23:59:59.99'
 , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo);

ALTER TABLE WebsiteUserInfo
SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.WebsiteUserInfoHistory));
GO

CREATE CLUSTERED COLUMNSTORE INDEX IX_WebsiteUserInfoHistory
ON dbo.WebsiteUserInfoHistory
WITH (DROP_EXISTING = ON);

```

## Step 2: Run your workload regularly

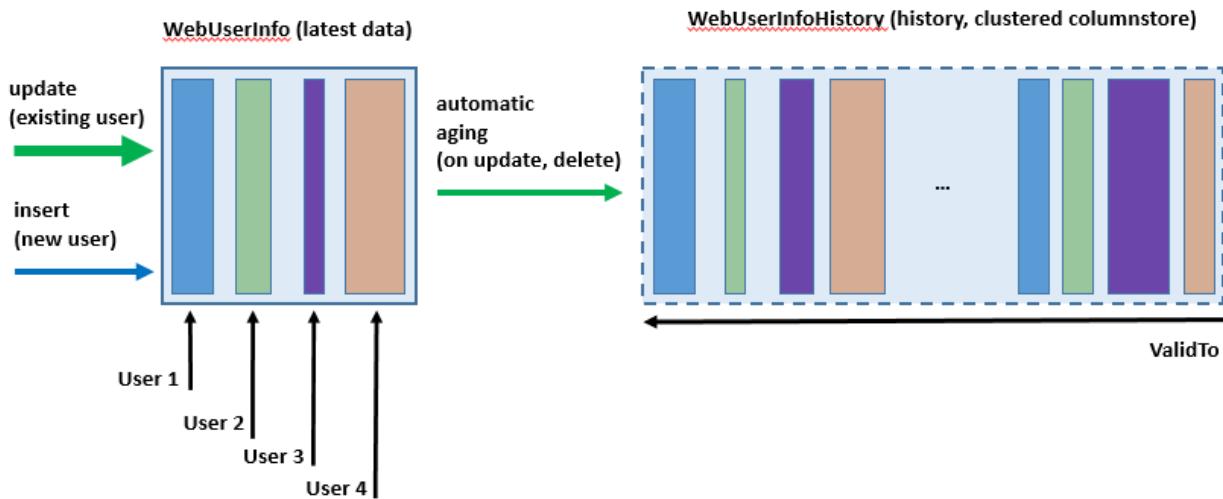
The main advantage of Temporal Tables is that you do not need to change or adjust your website in any way to

perform change tracking. Once created, Temporal Tables transparently persist previous row versions every time you perform modifications on your data.

In order to leverage automatic change tracking for this particular scenario, let's just update column **PagesVisited** every time a user ends their session on the website:

```
UPDATE WebsiteUserInfo SET [PagesVisited] = 5
WHERE [UserID] = 1;
```

It is important to notice that the update query doesn't need to know the exact time when the actual operation occurred nor how historical data will be preserved for future analysis. Both aspects are automatically handled by the Azure SQL Database. The following diagram illustrates how history data is being generated on every update.



## Step 3: Perform historical data analysis

Now when temporal system-versioning is enabled, historical data analysis is just one query away from you. In this article, we will provide a few examples that address common analysis scenarios - to learn all details, explore various options introduced with the [FOR SYSTEM\\_TIME](#) clause.

To see the top 10 users ordered by the number of visited web pages as of an hour ago, run this query:

```
DECLARE @hourAgo datetime2 = DATEADD(HOUR, -1, SYSUTCDATETIME());
SELECT TOP 10 * FROM dbo.WebsiteUserInfo FOR SYSTEM_TIME AS OF @hourAgo
ORDER BY PagesVisited DESC
```

You can easily modify this query to analyze the site visits as of a day ago, a month ago or at any point in the past you wish.

To perform basic statistical analysis for the previous day, use the following example:

```
DECLARE @twoDaysAgo datetime2 = DATEADD(DAY, -2, SYSUTCDATETIME());
DECLARE @aDayAgo datetime2 = DATEADD(DAY, -1, SYSUTCDATETIME());

SELECT UserID, SUM (PagesVisited) as TotalVisitedPages, AVG (PagesVisited) as AverageVisitedPages,
MAX (PagesVisited) AS MaxVisitedPages, MIN (PagesVisited) AS MinVisitedPages,
STDEV (PagesVisited) as StDevViistedPages
FROM dbo.WebsiteUserInfo
FOR SYSTEM_TIME BETWEEN @twoDaysAgo AND @aDayAgo
GROUP BY UserId
```

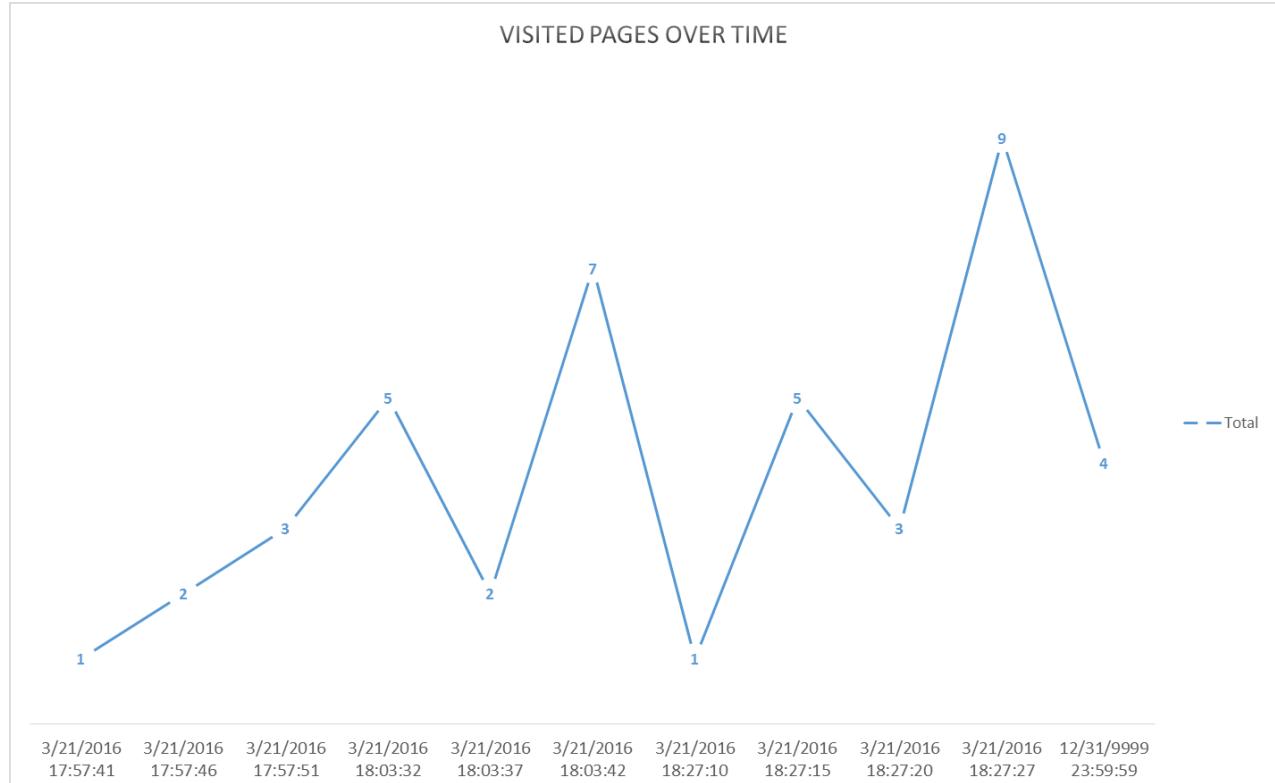
To search for activities of a specific user, within a period of time, use the **CONTAINED IN** clause:

```

DECLARE @hourAgo datetime2 = DATEADD(HOUR, -1, SYSUTCDATETIME());
DECLARE @twoHoursAgo datetime2 = DATEADD(HOUR, -2, SYSUTCDATETIME());
SELECT * FROM dbo.WebsiteUserInfo
FOR SYSTEM_TIME CONTAINED IN (@twoHoursAgo, @hourAgo)
WHERE [UserID] = 1;

```

Graphic visualization is especially convenient for temporal queries as you can show trends and usage patterns in an intuitive way very easily:



## Evolving table schema

Typically, you will need to change the temporal table schema while you are doing app development. For that, simply run regular ALTER TABLE statements and Azure SQL Database will appropriately propagate changes to the history table. The following script shows how you can add additional attribute for tracking:

```

/*Add new column for tracking source IP address*/
ALTER TABLE dbo.WebsiteUserInfo
ADD [IPAddress] varchar(128) NOT NULL CONSTRAINT DF_Address DEFAULT 'N/A';

```

Similarly, you can change column definition while your workload is active:

```

/*Increase the length of name column*/
ALTER TABLE dbo.WebsiteUserInfo
ALTER COLUMN UserName nvarchar(256) NOT NULL;

```

Finally, you can remove a column that you do not need anymore.

```

/*Drop unnecessary column */
ALTER TABLE dbo.WebsiteUserInfo
DROP COLUMN TemporaryColumn;

```

Alternatively, use latest [SSDT](#) to change temporal table schema while you are connected to the database (online

mode) or as part of the database project (offline mode).

## Controlling retention of historical data

With system-versioned temporal tables, the history table may increase the database size more than regular tables. A large and ever-growing history table can become an issue both due to pure storage costs as well as imposing a performance tax on temporal querying. Hence, developing a data retention policy for managing data in the history table is an important aspect of planning and managing the lifecycle of every temporal table. With Azure SQL Database, you have the following approaches for managing historical data in the temporal table:

- [Table Partitioning](#)
- [Custom Cleanup Script](#)

## Next steps

- For more information on Temporal Tables, see check out [Temporal Tables](#).
- Visit Channel 9 to hear a [real customer temporal implementation success story](#) and watch a [live temporal demonstration](#).

# Automate management tasks using database jobs

2/7/2020 • 14 minutes to read • [Edit Online](#)

Azure SQL Database enables you to create and schedule jobs that could be periodically executed against one or many databases to run T-SQL queries and perform maintenance tasks. Every job logs the status of execution and also automatically retries the operations if any failure occurs. You can define target database or groups of Azure SQL databases where the job will be executed, and also define schedules for running a job. A job handles the task of logging in to the target database. You also define, maintain, and persist Transact-SQL scripts to be executed across a group of Azure SQL databases.

## When to use automated jobs

There are several scenarios when you could use job automation:

- Automate management tasks and schedule them to run every weekday, after hours, etc.
  - Deploy schema changes, credentials management, performance data collection or tenant (customer) telemetry collection.
  - Update reference data (information common across all databases), load data from Azure Blob storage.
  - Rebuild indexes to improve query performance. Configure jobs to execute across a collection of databases on a recurring basis, such as during off-peak hours.
  - Collect query results from a set of databases into a central table on an on-going basis. Performance queries can be continually executed and configured to trigger additional tasks to be executed.
- Collect data for reporting
  - Aggregate data from a collection of Azure SQL databases into a single destination table.
  - Execute longer running data processing queries across a large set of databases, for example the collection of customer telemetry. Results are collected into a single destination table for further analysis.
- Data movements
  - Create jobs that replicate changes made in your databases to other databases or collect updates made in remote databases and apply changes in the database.
  - Create jobs that load data from or to your databases using SQL Server Integration Services (SSIS).

## Overview

The following job scheduling technologies are available in Azure SQL Database:

- **SQL Agent Jobs** are classic and battle-tested SQL Server job scheduling component that is available in Managed Instance. SQL Agent Jobs are not available in Azure SQL single databases.
- **Elastic Database Jobs (preview)** are Job Scheduling services that execute custom jobs on one or many Azure SQL Databases.

It is worth noting a couple of differences between SQL Agent (available on-premises and as part of SQL Database Managed Instance), and the Database Elastic Job agent (available for single databases in Azure SQL database and databases in SQL Data Warehouse).

ELASTIC JOBS

SQL AGENT

	<b>ELASTIC JOBS</b>	<b>SQL AGENT</b>
Scope	<p>Any number of Azure SQL databases and/or data warehouses in the same Azure cloud as the job agent. Targets can be in different SQL Database servers, subscriptions, and/or regions.</p> <p>Target groups can be composed of individual databases or data warehouses, or all databases in a server, pool, or shardmap (dynamically enumerated at job runtime).</p>	Any individual database in the same SQL Server instance as the SQL agent.
Supported APIs and Tools	Portal, PowerShell, T-SQL, Azure Resource Manager	T-SQL, SQL Server Management Studio (SSMS)

## SQL Agent Jobs

SQL Agent Jobs are a specified series of T-SQL scripts against your database. Use jobs to define an administrative task that can be run one or more times and monitored for success or failure. A job can run on one local server or on multiple remote servers. SQL Agent Jobs are an internal Database Engine component that is executed within the Managed Instance service. There are several key concepts in SQL Agent Jobs:

- **Job steps** set of one or many steps that should be executed within the job. For every job step you can define retry strategy and the action that should happen if the job step succeeds or fails.
- **Schedules** define when the job should be executed.
- **Notifications** enable you to define rules that will be used to notify operators via email once the job completes.

### Job steps

SQL Agent Job steps are sequences of actions that SQL Agent should execute. Every step has the following step that should be executed if the step succeeds or fails, number of retries in a case of failure. SQL Agent enables you to create different types of job steps, such as Transact-SQL job step that executes a single Transact-SQL batch against the database, or OS command/PowerShell steps that can execute custom OS script, SSIS job steps enable you to load data using SSIS runtime, or [replication](#) steps that can publish changes from your database to other databases.

[Transactional replication](#) is a Database Engine feature that enables you to publish the changes made on one or multiple tables in one database and publish/distribute them to a set of subscriber databases. Publishing of the changes is implemented using the following SQL Agent job step types:

- Transaction-log reader.
- Snapshot.
- Distributor.

Other types of job steps are not currently supported, including:

- Merge replication job step is not supported.
- Queue Reader is not supported.
- Analysis Services are not supported

### Job schedules

A schedule specifies when a job runs. More than one job can run on the same schedule, and more than one schedule can apply to the same job. A schedule can define the following conditions for the time when a job runs:

- Whenever Instance is restarted (or when SQL Server Agent starts). Job is activated after every failover.

- One time, at a specific date and time, which is useful for delayed execution of some job.
- On a recurring schedule.

#### **NOTE**

Managed Instance currently does not enable you to start a job when the instance is "idle".

### **Job notifications**

SQL Agent Jobs enable you to get notifications when the job finishes successfully or fails. You can receive notifications via email.

First, you would need to set up the email account that will be used to send the email notifications and assign the account to the email profile called `AzureManagedInstance_dbmail_profile`, as shown in the following sample:

```
-- Create a Database Mail account
EXECUTE msdb.dbo.sysmail_add_account_sp
 @account_name = 'SQL Agent Account',
 @description = 'Mail account for Azure SQL Managed Instance SQL Agent system.',
 @email_address = '$(loginEmail)',
 @display_name = 'SQL Agent Account',
 @mailserver_name = '$(mailserver)' ,
 @username = '$(loginEmail)' ,
 @password = '$(password)'

-- Create a Database Mail profile
EXECUTE msdb.dbo.sysmail_add_profile_sp
 @profile_name = 'AzureManagedInstance_dbmail_profile',
 @description = 'E-mail profile used for messages sent by Managed Instance SQL Agent.' ;

-- Add the account to the profile
EXECUTE msdb.dbo.sysmail_add_profileaccount_sp
 @profile_name = 'AzureManagedInstance_dbmail_profile',
 @account_name = 'SQL Agent Account',
 @sequence_number = 1;
```

You would also need to enable Database Mail on Managed Instance:

```
GO
EXEC sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
EXEC sp_configure 'Database Mail XPs', 1;
GO
RECONFIGURE
```

You can notify the operator that something happened with your SQL Agent jobs. An operator defines contact information for an individual responsible for the maintenance of one or more Managed Instances. Sometimes, operator responsibilities are assigned to one individual. In systems with multiple Managed Instance or SQL Servers, many individuals can share operator responsibilities. An operator does not contain security information, and does not define a security principal.

You can create operators using SSMS or the Transact-SQL script shown in the following example:

```
EXEC msdb.dbo.sp_add_operator
 @name=N'Mihajlo Pupun',
 @enabled=1,
 @email_address=N'mihajlo.pupin@contoso.com'
```

You can modify any job and assign operators that will be notified via email if the job completes, fails, or succeeds using SSMS or the following Transact-SQL script:

```
EXEC msdb.dbo.sp_update_job @job_name=N'Load data using SSIS',
 @notify_level_email=3, -- Options are: 1 on succeed, 2 on failure, 3 on complete
 @notify_email_operator_name=N'Mihajlo Pupun'
```

## SQL Agent Job Limitations

Some of the SQL Agent features that are available in SQL Server are not supported in Managed Instance:

- SQL Agent settings are read only. Procedure `sp_set_agent_properties` is not supported in Managed Instance.
- Enabling/disabling SQL Agent is currently not supported in Managed Instance. SQL Agent is always running.
- Notifications are partially supported
  - Pager is not supported.
  - NetSend is not supported.
  - Alerts are not supported.
- Proxies are not supported.
- Eventlog is not supported.

For information about SQL Server Agent, see [SQL Server Agent](#).

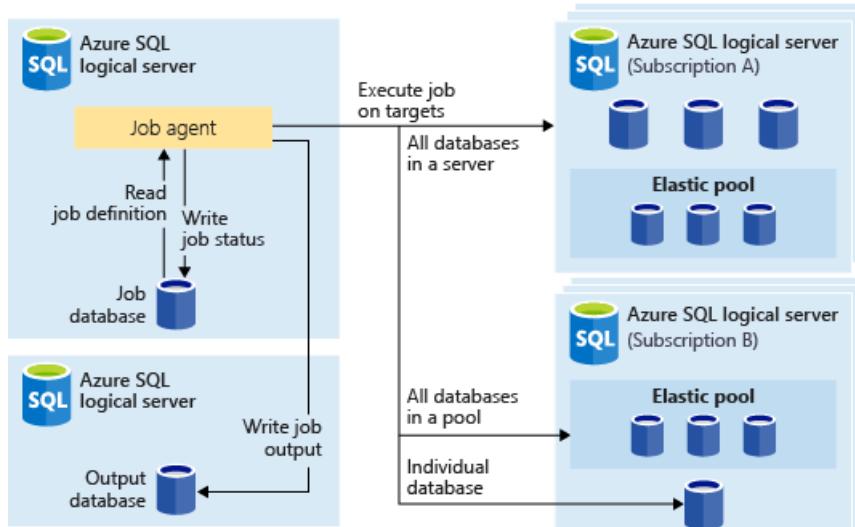
## Elastic Database Jobs (preview)

**Elastic Database Jobs** provide the ability to run one or more T-SQL scripts in parallel, across a large number of databases, on a schedule or on-demand.

**Run jobs against any combination of databases:** one or more individual databases, all databases on a server, all databases in an elastic pool, or shardmap, with the added flexibility to include or exclude any specific database.

**Jobs can run across multiple servers, multiple pools, and can even run against databases in different subscriptions.** Servers and pools are dynamically enumerated at runtime, so jobs run against all databases that exist in the target group at the time of execution.

The following image shows a job agent executing jobs across the different types of target groups:



## Elastic Job components

COMPONENT	DESCRIPTION (ADDITIONAL DETAILS ARE BELOW THE TABLE)
Elastic Job agent	The Azure resource you create to run and manage Jobs.
Job database	An Azure SQL database the job agent uses to store job related data, job definitions, etc.
Target group	The set of servers, pools, databases, and shard maps to run a job against.
Job	A job is a unit of work that is composed of one or more <a href="#">job steps</a> . Job steps specify the T-SQL script to run, as well as other details required to execute the script.

### Elastic Job agent

An Elastic Job agent is the Azure resource for creating, running, and managing jobs. The Elastic Job agent is an Azure resource you create in the portal ([PowerShell](#) and REST are also supported).

Creating an **Elastic Job agent** requires an existing SQL database. The agent configures this existing database as the [Job database](#).

The Elastic Job agent is free. The job database is billed at the same rate as any SQL database.

### Job database

The *Job database* is used for defining jobs and tracking the status and history of job executions. The *Job database* is also used to store agent metadata, logs, results, job definitions, and also contains many useful stored procedures and other database objects for creating, running, and managing jobs using T-SQL.

For the current preview, an existing Azure SQL database (S0 or higher) is required to create an Elastic Job agent.

The *Job database* doesn't literally need to be new, but should be a clean, empty, S0 or higher service objective. The recommended service objective of the *Job database* is S1 or higher, but the optimal choice depends on the performance needs of your job(s): the number of job steps, the number of job targets, and how frequently jobs are run. For example, an S0 database might be sufficient for a job agent that runs few jobs an hour targeting less than ten databases, but running a job every minute might not be fast enough with an S0 database, and a higher service tier might be better.

If operations against the job database are slower than expected, [monitor](#) database performance and the resource utilization in the job database during periods of slowness using Azure portal or the [sys.dm\\_db\\_resource\\_stats](#) DMV. If utilization of a resource, such as CPU, Data IO, or Log Write approaches 100% and correlates with periods of slowness, consider incrementally scaling the database to higher service objectives (either in the [DTU model](#) or in the [vCore model](#)) until job database performance is sufficiently improved.

#### Job database permissions

During job agent creation, a schema, tables, and a role called *jobs\_reader* are created in the *Job database*. The role is created with the following permission and is designed to give administrators finer access control for job monitoring:

ROLE NAME	'JOBS' SCHEMA PERMISSIONS	'JOBS_INTERNAL' SCHEMA PERMISSIONS
<b>jobs_reader</b>	SELECT	None

## IMPORTANT

Consider the security implications before granting access to the *Job database* as a database administrator. A malicious user with permissions to create or edit jobs could create or edit a job that uses a stored credential to connect to a database under the malicious user's control, which could allow the malicious user to determine the credential's password.

### Target group

A *target group* defines the set of databases a job step will execute on. A target group can contain any number and combination of the following:

- **SQL Database server** - if a server is specified, all databases that exist in the server at the time of the job execution are part of the group. The master database credential must be provided so that the group can be enumerated and updated prior to job execution.
- **Elastic pool** - if an elastic pool is specified, all databases that are in the elastic pool at the time of the job execution are part of the group. As for a server, the master database credential must be provided so that the group can be updated prior to the job execution.
- **Single database** - specify one or more individual databases to be part of the group.
- **Shardmap** - databases of a shardmap.

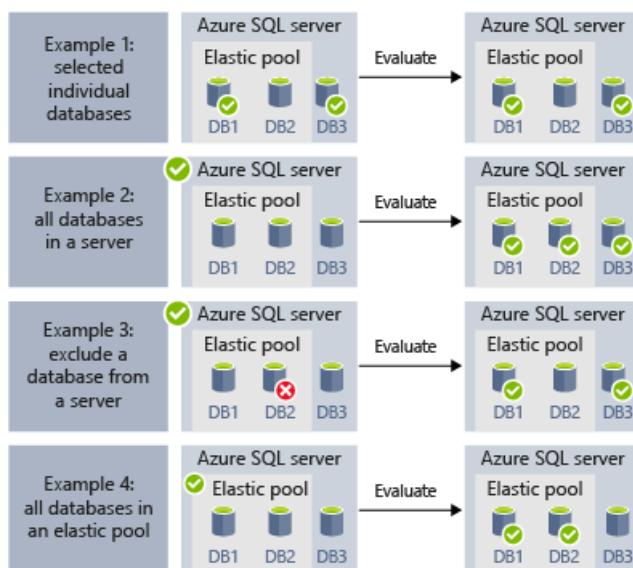
### TIP

At the moment of job execution, *dynamic enumeration* re-evaluates the set of databases in target groups that include servers or pools. Dynamic enumeration ensures that **jobs run across all databases that exist in the server or pool at the time of job execution**. Re-evaluating the list of databases at runtime is specifically useful for scenarios where pool or server membership changes frequently.

Pools and single databases can be specified as included or excluded from the group. This enables creating a target group with any combination of databases. For example, you can add a server to a target group, but exclude specific databases in an elastic pool (or exclude an entire pool).

A target group can include databases in multiple subscriptions, and across multiple regions. Note that cross-region executions have higher latency than executions within the same region.

The following examples show how different target group definitions are dynamically enumerated at the moment of job execution to determine which databases the job will run:

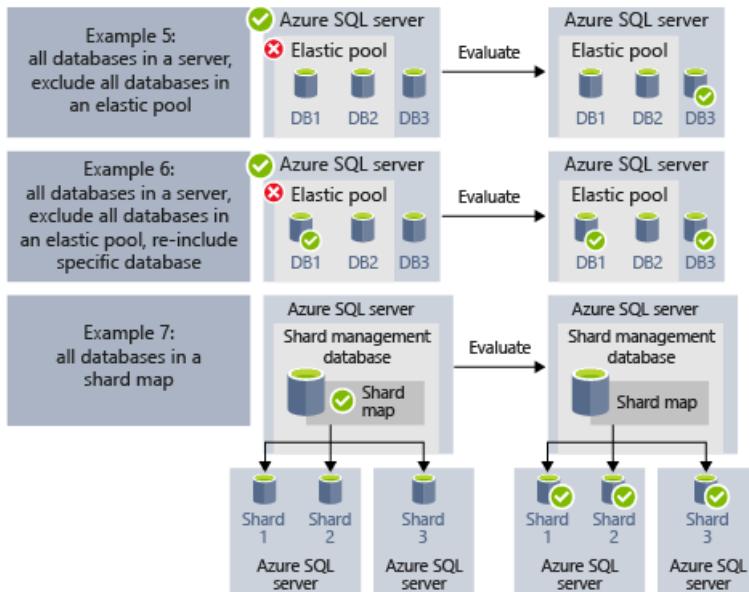


**Example 1** shows a target group that consists of a list of individual databases. When a job step is executed using this target group, the job step's action will be executed in each of those databases.

**Example 2** shows a target group that contains an Azure SQL Server as a target. When a job step is executed using this target group, the server is dynamically enumerated to determine the list of databases that are currently in the server. The job step's action will be executed in each of those databases.

**Example 3** shows a similar target group as *Example 2*, but an individual database is specifically excluded. The job step's action will *not* be executed in the excluded database.

**Example 4** shows a target group that contains an elastic pool as a target. Similar to *Example 2*, the pool will be dynamically enumerated at job run time to determine the list of databases in the pool.



**Example 5** and **Example 6** show advanced scenarios where Azure SQL Servers, elastic pools, and databases can be combined using include and exclude rules.

**Example 7** shows that the shards in a shard map can also be evaluated at job run time.

#### NOTE

The Job database itself can be the target of a job. In this scenario, the Job database is treated just like any other target database. The job user must be created and granted sufficient permissions in the Job database, and the database scoped credential for the job user must also exist in the Job database, just like it does for any other target database.

#### Job

A *job* is a unit of work that is executed on a schedule or as a one-time job. A job consists of one or more *job steps*.

#### Job step

Each job step specifies a T-SQL script to execute, one or more target groups to run the T-SQL script against, and the credentials the job agent needs to connect to the target database. Each job step has customizable timeout and retry policies, and can optionally specify output parameters.

#### Job output

The outcome of a job's steps on each target database are recorded in detail, and script output can be captured to a specified table. You can specify a database to save any data returned from a job.

#### Job history

Job execution history is stored in the *Job database*. A system cleanup job purges execution history that is older than 45 days. To remove history less than 45 days old, call the **sp\_purge\_history** stored procedure in the *Job database*.

#### Agent performance, capacity, and limitations

Elastic Jobs use minimal compute resources while waiting for long-running jobs to complete.

Depending on the size of the target group of databases and the desired execution time for a job (number of

concurrent workers), the agent requires different amounts of compute and performance of the *Job database* (the more targets and the higher number of jobs, the higher the amount of compute required).

Currently, the preview is limited to 100 concurrent jobs.

#### **Prevent jobs from reducing target database performance**

To ensure resources aren't overburdened when running jobs against databases in a SQL elastic pool, jobs can be configured to limit the number of databases a job can run against at the same time.

## Next steps

- [What is SQL Server Agent](#)
- [How to create and manage elastic jobs](#)
- [Create and manage Elastic Jobs using PowerShell](#)
- [Create and manage Elastic Jobs using Transact-SQL \(T-SQL\)](#)

# Tutorial: Design a relational database in a single database within Azure SQL Database using SSMS

12/23/2019 • 8 minutes to read • [Edit Online](#)

Azure SQL database is a relational database-as-a-service (DBaaS) in the Microsoft Cloud (Azure). In this tutorial, you learn how to use the Azure portal and [SQL Server Management Studio \(SSMS\)](#) to:

- Create a single database using the Azure portal\*
- Set up a server-level IP firewall rule using the Azure portal
- Connect to the database with SSMS
- Create tables with SSMS
- Bulk load data with BCP
- Query data with SSMS

\*If you don't have an Azure subscription, [create a free account](#) before you begin.

## TIP

The following Microsoft Learn module helps you learn for free how to [Develop and configure an ASP.NET application that queries an Azure SQL Database](#), including the creation of a simple database.

## NOTE

For the purpose of this tutorial, we are using a single database. You could also use a pooled database in an elastic pool or an instance database in a managed instance. For connectivity to a managed instance, see these managed instance quickstarts: [Quickstart: Configure Azure VM to connect to an Azure SQL Database Managed Instance](#) and [Quickstart: Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises](#).

## Prerequisites

To complete this tutorial, make sure you've installed:

- [SQL Server Management Studio](#) (latest version)
- [BCP and SQLCMD](#) (latest version)

## Sign in to the Azure portal

Sign in to the [Azure portal](#).

## Create a blank single database

A single database in Azure SQL Database is created with a defined set of compute and storage resources. The database is created within an [Azure resource group](#) and is managed using an [database server](#).

Follow these steps to create a blank single database.

1. On the Azure portal menu or from the **Home** page, select **Create a resource**.
2. On the **New** page, select **Databases** in the Azure Marketplace section, and then click **SQL Database** in

the **Featured** section.

The screenshot shows the Azure Marketplace 'New' page. On the left, there's a sidebar with categories like 'Get started', 'Recently created', 'AI + Machine Learning', 'Analytics', 'Blockchain', 'Compute', 'Containers', 'Databases' (which is highlighted with a red box), 'Developer Tools', 'DevOps', 'Identity', 'Integration', 'Internet of Things', 'Media', 'Mixed Reality', 'IT & Management Tools', 'Networking', 'Software as a Service (SaaS)', and 'Security'. On the right, under the 'Featured' heading, there are several service cards. One card for 'SQL Database' has a red box around it. Other cards include 'Azure SQL Managed Instance', 'SQL Data Warehouse', 'Azure Database for MariaDB', 'Couchbase Enterprise Edition (Hourly Pricing) (preview)', 'Azure Database for MySQL', 'Azure Database for PostgreSQL', 'Azure Cosmos DB', and 'SQL Server 2017 Enterprise Windows Server 2016'.

3. Fill out the **SQL Database** form with the following information, as shown on the preceding image:

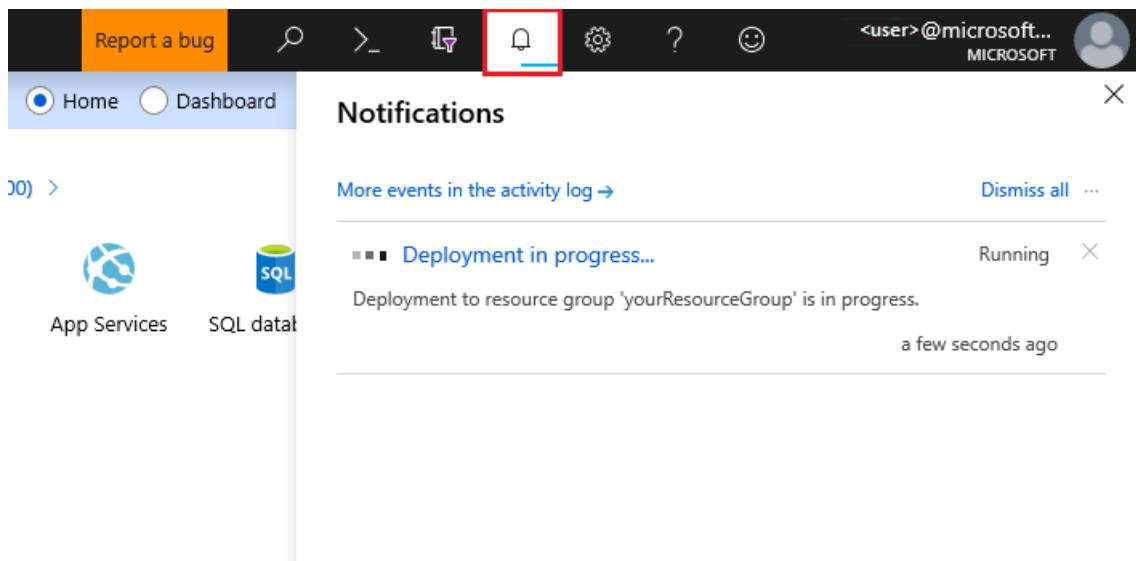
SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Database name</b>	<i>yourDatabase</i>	For valid database names, see <a href="#">Database identifiers</a> .
<b>Subscription</b>	<i>yourSubscription</i>	For details about your subscriptions, see <a href="#">Subscriptions</a> .
<b>Resource group</b>	<i>yourResourceGroup</i>	For valid resource group names, see <a href="#">Naming rules and restrictions</a> .
<b>Select source</b>	Blank database	Specifies that a blank database should be created.

4. Click **Server** to use an existing database server or create and configure a new database server. Either select an existing server or click **Create a new server** and fill out the **New server** form with the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Server name</b>	Any globally unique name	For valid server names, see <a href="#">Naming rules and restrictions</a> .
<b>Server admin login</b>	Any valid name	For valid login names, see <a href="#">Database identifiers</a> .
<b>Password</b>	Any valid password	Your password must have at least eight characters and must use characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters.
<b>Location</b>	Any valid location	For information about regions, see <a href="#">Azure Regions</a> .

The screenshot shows the 'New server' configuration dialog. On the left, there's a sidebar with a '+ Create a new server' button and a list of existing servers ('existingserver', 'West US 2', 'yourR...'). The main area contains fields for 'Server name' (set to '<yourServer>'), 'Server admin login' (set to '<yourUser>'), 'Password' (a masked password), 'Confirm password' (another masked password), and 'Location' (set to 'West US 2'). A checkbox for 'Allow Azure services to access server' is checked. At the bottom is a 'Select' button.

5. Click **Select**.
6. Click **Pricing tier** to specify the service tier, the number of DTUs or vCores, and the amount of storage. You may explore the options for the number of DTUs/vCores and storage that is available to you for each service tier.  
After selecting the service tier, the number of DTUs or vCores, and the amount of storage, click **Apply**.
7. Enter a **Collation** for the blank database (for this tutorial, use the default value). For more information about collations, see [Collations](#)
8. Now that you've completed the **SQL Database** form, click **Create** to provision the single database. This step may take a few minutes.
9. On the toolbar, click **Notifications** to monitor the deployment process.



## Create a server-level IP firewall rule

The SQL Database service creates an IP firewall at the server-level. This firewall prevents external applications and tools from connecting to the server and any databases on the server unless a firewall rule allows their IP through the firewall. To enable external connectivity to your single database, you must first add an IP firewall rule for your IP address (or IP address range). Follow these steps to create a [SQL Database server-level IP firewall rule](#).

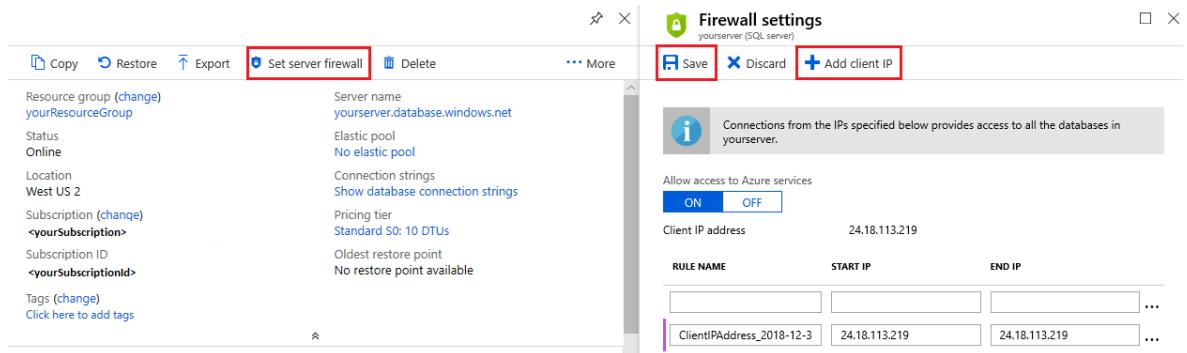
### IMPORTANT

The SQL Database service communicates over port 1433. If you are trying to connect to this service from within a corporate network, outbound traffic over port 1433 may not be allowed by your network's firewall. If so, you cannot connect to your single database unless your administrator opens port 1433.

1. After the deployment completes, select **SQL databases** from the Azure portal menu or search for and select *SQL databases* from any page.
2. Select *yourDatabase* on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified **Server name** (such as `contosodatabaseserver01.database.windows.net`) and provides options for further configuration.

The screenshot shows the Azure portal's SQL database overview page for 'contosodatabase14'. The left sidebar lists various options like Overview, Activity log, Tags, etc. The main pane shows details such as Resource group (RG01), Status (Online), Location (West US 2), Subscription (Contoso Ltd Subscription), and Pricing tier (General Purpose: Gen5, 2 vCores). The 'Server name' field is highlighted with a red box and contains the value 'contosodatabaseserver01.database.windows.net'. The 'Set server firewall' button is also highlighted with a red box.

3. Copy this fully qualified server name for use to connect to your server and databases from SQL Server Management Studio.
4. Click **Set server firewall** on the toolbar. The **Firewall settings** page for the SQL Database server opens.



5. Click **Add client IP** on the toolbar to add your current IP address to a new IP firewall rule. An IP firewall rule can open port 1433 for a single IP address or a range of IP addresses.
6. Click **Save**. A server-level IP firewall rule is created for your current IP address opening port 1433 on the SQL Database server.
7. Click **OK** and then close the **Firewall settings** page.

Your IP address can now pass through the IP firewall. You can now connect to your single database using SQL Server Management Studio or another tool of your choice. Be sure to use the server admin account you created previously.

#### IMPORTANT

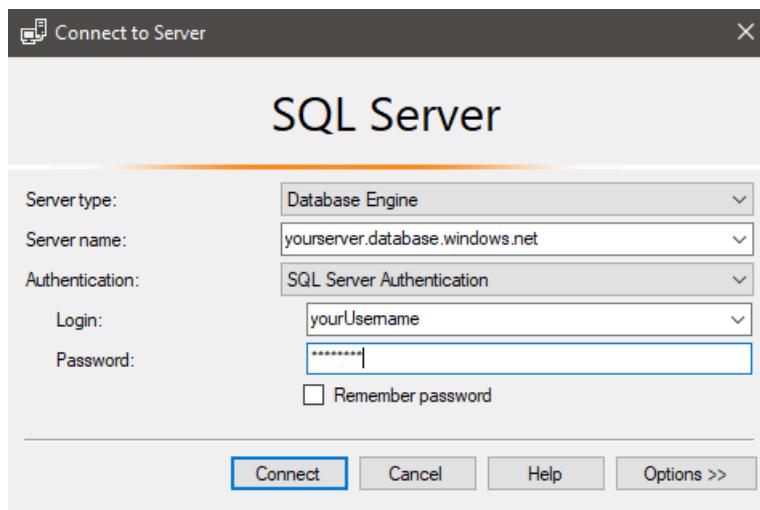
By default, access through the SQL Database IP firewall is enabled for all Azure services. Click **OFF** on this page to disable for all Azure services.

## Connect to the database

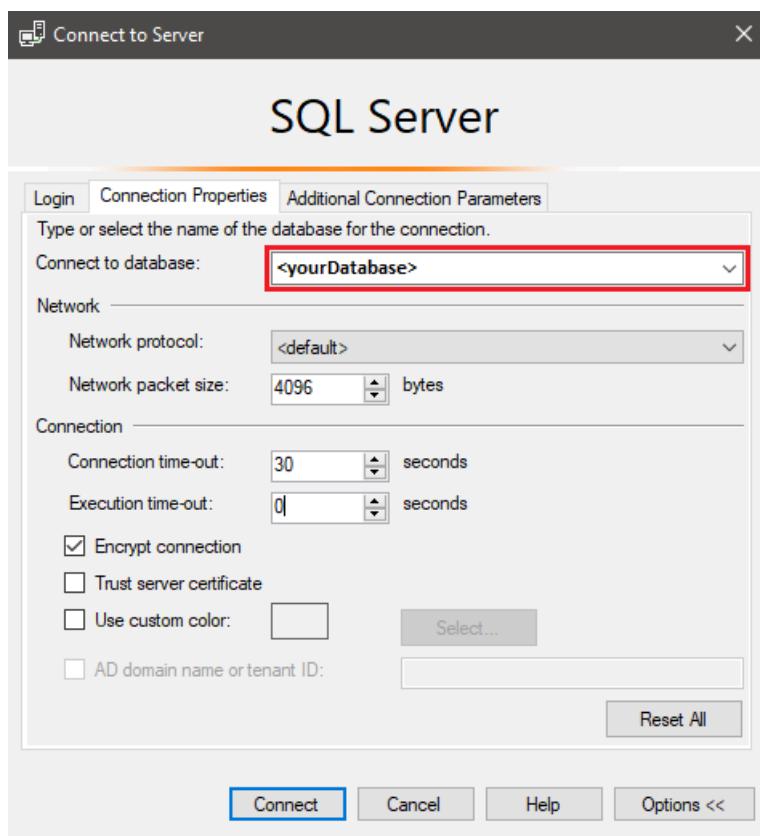
Use [SQL Server Management Studio](#) to establish a connection to your single database.

1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

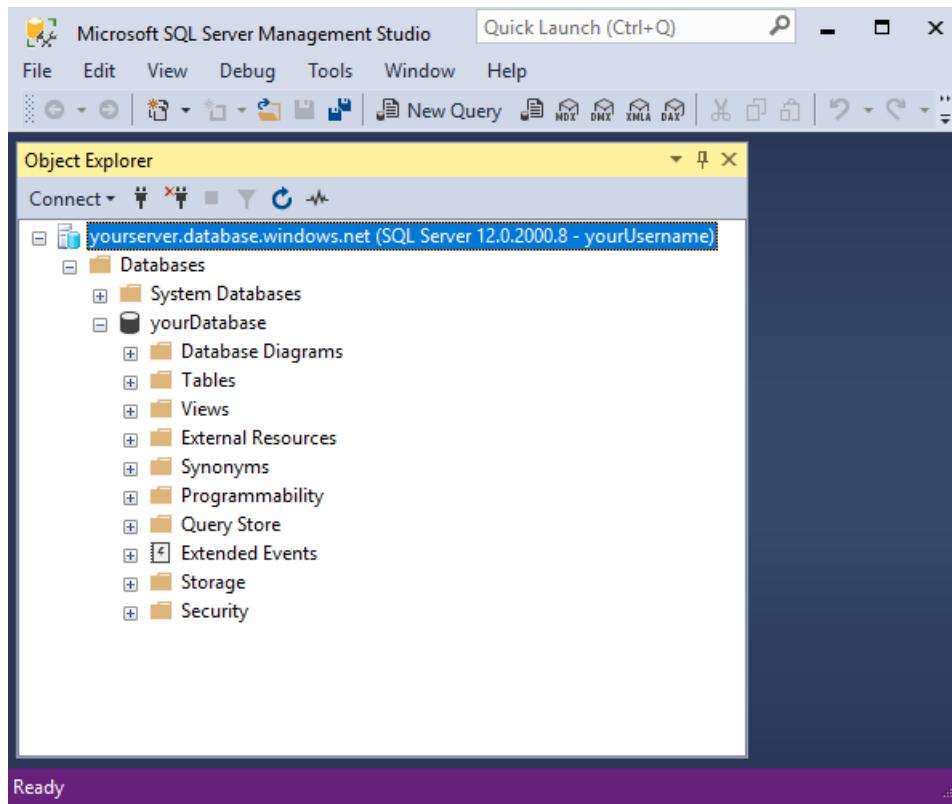
SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Server type</b>	Database engine	This value is required.
<b>Server name</b>	The fully qualified server name	For example, <i>yourserver.database.windows.net</i> .
<b>Authentication</b>	SQL Server Authentication	SQL Authentication is the only authentication type that we've configured in this tutorial.
<b>Login</b>	The server admin account	The account that you specified when you created the server.
<b>Password</b>	The password for your server admin account	The password that you specified when you created the server.



3. Click **Options** in the **Connect to server** dialog box. In the **Connect to database** section, enter *yourDatabase* to connect to this database.



4. Click **Connect**. The **Object Explorer** window opens in SSMS.
5. In **Object Explorer**, expand **Databases** and then expand *yourDatabase* to view the objects in the sample database.



## Create tables in your database

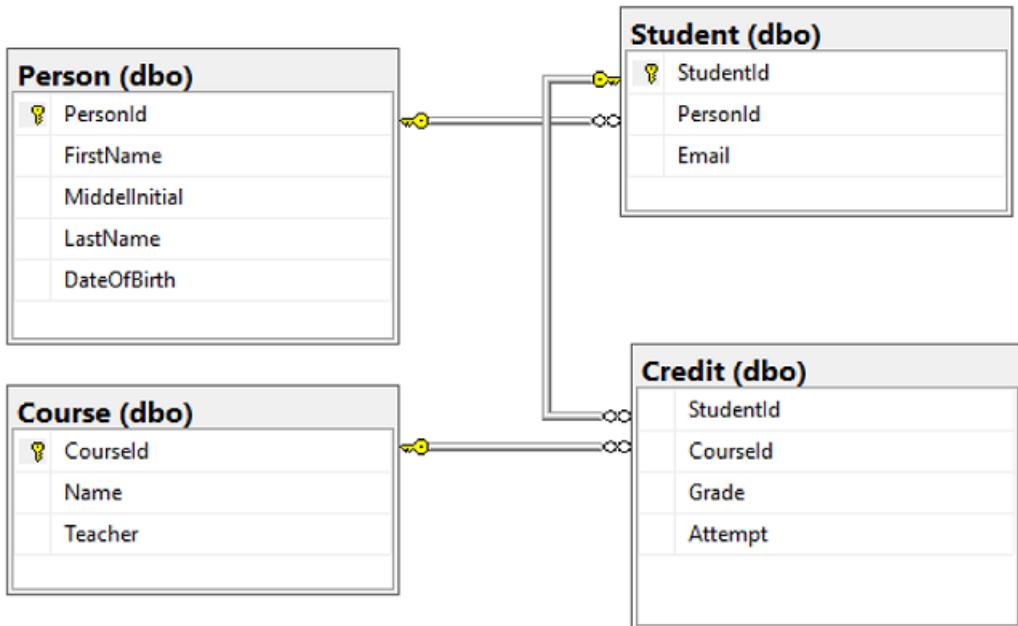
Create a database schema with four tables that model a student management system for universities using [Transact-SQL](#):

- Person
- Course
- Student
- Credit

The following diagram shows how these tables are related to each other. Some of these tables reference columns in other tables. For example, the *Student* table references the *PersonId* column of the *Person* table. Study the diagram to understand how the tables in this tutorial are related to one another. For an in-depth look at how to create effective database tables, see [Create effective database tables](#). For information about choosing data types, see [Data types](#).

### NOTE

You can also use the [table designer](#) in SQL Server Management Studio to create and design your tables.



1. In **Object Explorer**, right-click *yourDatabase* and select **New Query**. A blank query window opens that is connected to your database.
2. In the query window, execute the following query to create four tables in your database:

```

-- Create Person table
CREATE TABLE Person
(
 PersonId INT IDENTITY PRIMARY KEY,
 FirstName NVARCHAR(128) NOT NULL,
 MiddleInitial NVARCHAR(10),
 LastName NVARCHAR(128) NOT NULL,
 DateOfBirth DATE NOT NULL
)

-- Create Student table
CREATE TABLE Student
(
 StudentId INT IDENTITY PRIMARY KEY,
 PersonId INT REFERENCES Person (PersonId),
 Email NVARCHAR(256)
)

-- Create Course table
CREATE TABLE Course
(
 CourseId INT IDENTITY PRIMARY KEY,
 Name NVARCHAR(50) NOT NULL,
 Teacher NVARCHAR(256) NOT NULL
)

-- Create Credit table
CREATE TABLE Credit
(
 StudentId INT REFERENCES Student (StudentId),
 CourseId INT REFERENCES Course (CourseId),
 Grade DECIMAL(5,2) CHECK (Grade <= 100.00),
 Attempt TINYINT,
 CONSTRAINT [UQ_studentgrades] UNIQUE CLUSTERED
 (
 StudentId, CourseId, Grade, Attempt
)
)

```

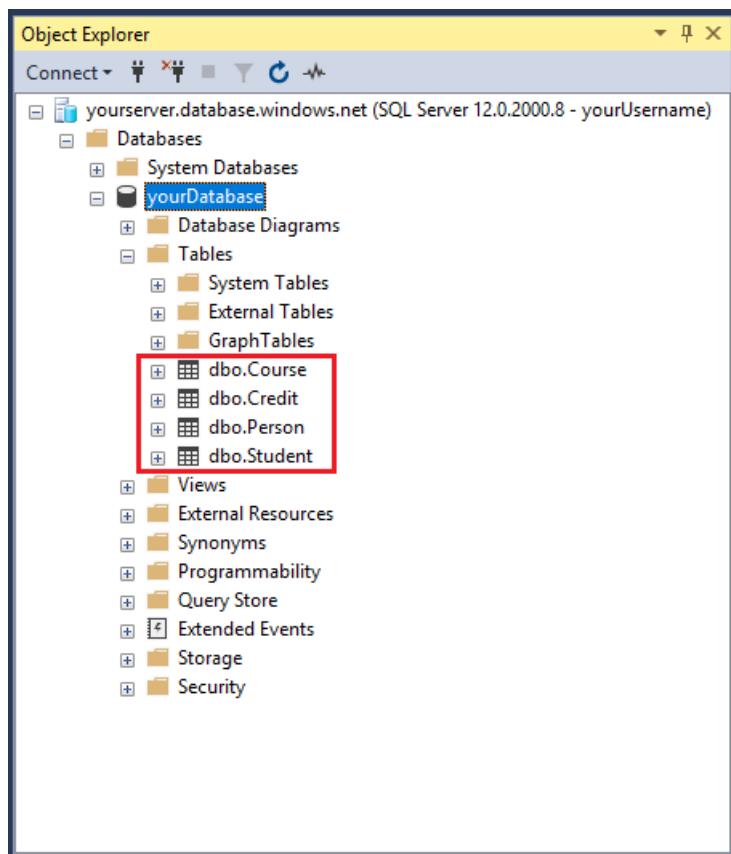
The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, under 'yourDatabase', the 'Tables' node is expanded, showing four tables: Course, Credit, Person, and Student. In the center pane, a query window displays two CREATE TABLE statements:

```
-- Create Person table
CREATE TABLE Person
(
 PersonId INT IDENTITY PRIMARY KEY,
 FirstName NVARCHAR(128) NOT NULL,
 MiddleInitial NVARCHAR(10),
 LastName NVARCHAR(128) NOT NULL,
 DateOfBirth DATE NOT NULL
)

-- Create Student table
CREATE TABLE Student
(
 StudentId INT IDENTITY PRIMARY KEY,
 PersonId INT REFERENCES Person (PersonId),
 Email NVARCHAR(256)
)
```

The status bar at the bottom indicates 'Commands completed successfully.' and 'Query executed successfully.'

3. Expand the **Tables** node under *yourDatabase* in the **Object Explorer** to see the tables you created.



## Load data into the tables

1. Create a folder called *sampleData* in your Downloads folder to store sample data for your database.
2. Right-click the following links and save them into the *sampleData* folder.
  - [SampleCourseData](#)
  - [SamplePersonData](#)
  - [SampleStudentData](#)
  - [SampleCreditData](#)
3. Open a command prompt window and navigate to the *sampleData* folder.

4. Execute the following commands to insert sample data into the tables replacing the values for *server*, *database*, *user*, and *password* with the values for your environment.

```
bcp Course in SampleCourseData -S <server>.database.windows.net -d <database> -U <user> -P <password>
-q -c -t ","
bcp Person in SamplePersonData -S <server>.database.windows.net -d <database> -U <user> -P <password>
-q -c -t ","
bcp Student in SampleStudentData -S <server>.database.windows.net -d <database> -U <user> -P <password>
-q -c -t ","
bcp Credit in SampleCreditData -S <server>.database.windows.net -d <database> -U <user> -P <password>
-q -c -t ","
```

You have now loaded sample data into the tables you created earlier.

## Query data

Execute the following queries to retrieve information from the database tables. See [Write SQL queries](#) to learn more about writing SQL queries. The first query joins all four tables to find the students taught by 'Dominick Pope' who have a grade higher than 75%. The second query joins all four tables and finds the courses in which 'Noe Coleman' has ever enrolled.

1. In a SQL Server Management Studio query window, execute the following query:

```
-- Find the students taught by Dominick Pope who have a grade higher than 75%
SELECT person.FirstName, person.LastName, course.Name, credit.Grade
FROM Person AS person
 INNER JOIN Student AS student ON person.PersonId = student.PersonId
 INNER JOIN Credit AS credit ON student.StudentId = credit.StudentId
 INNER JOIN Course AS course ON credit.CourseId = course.courseId
WHERE course.Teacher = 'Dominick Pope'
 AND Grade > 75
```

2. In a query window, execute the following query:

```
-- Find all the courses in which Noe Coleman has ever enrolled
SELECT course.Name, course.Teacher, credit.Grade
FROM Course AS course
 INNER JOIN Credit AS credit ON credit.CourseId = course.CourseId
 INNER JOIN Student AS student ON student.StudentId = credit.StudentId
 INNER JOIN Person AS person ON person.PersonId = student.PersonId
WHERE person.FirstName = 'Noe'
 AND person.LastName = 'Coleman'
```

## Next steps

In this tutorial, you learned many basic database tasks. You learned how to:

- Create a single database
- Set up a server-level IP firewall rule
- Connect to the database with [SQL Server Management Studio \(SSMS\)](#)
- Create tables
- Bulk load data
- Query that data

Advance to the next tutorial to learn about designing a database using Visual Studio and C#.

[Design a relational database in a single database within Azure SQL Database C# and ADO.NET](#)



# Tutorial: Design a relational database in a single database within Azure SQL Database C# and ADO.NET

12/23/2019 • 9 minutes to read • [Edit Online](#)

Azure SQL Database is a relational database-as-a-service (DBaaS) in the Microsoft Cloud (Azure). In this tutorial, you learn how to use the Azure portal and ADO.NET with Visual Studio to:

- Create a single database using the Azure portal\*
- Set up a server-level IP firewall rule using the Azure portal
- Connect to the database with ADO.NET and Visual Studio
- Create tables with ADO.NET
- Insert, update, and delete data with ADO.NET
- Query data ADO.NET

\*If you don't have an Azure subscription, [create a free account](#) before you begin.

## TIP

The following Microsoft Learn module helps you learn for free how to [Develop and configure an ASP.NET application that queries an Azure SQL Database](#), including the creation of a simple database.

## Prerequisites

An installation of [Visual Studio 2019](#) or later.

## Create a blank single database

A single database in Azure SQL Database is created with a defined set of compute and storage resources. The database is created within an [Azure resource group](#) and is managed using an [database server](#).

Follow these steps to create a blank single database.

1. Click **Create a resource** in the upper left-hand corner of the Azure portal.
2. On the **New** page, select **Databases** in the Azure Marketplace section, and then click **SQL Database** in the **Featured** section.

The screenshot shows the Azure Marketplace 'New' page. The 'Databases' category is highlighted with a red box. Within this category, the 'SQL Database' item is also highlighted with a red box. Other items in the list include 'Azure SQL Managed Instance', 'SQL Data Warehouse', 'Azure Database for MariaDB', 'Couchbase Enterprise Edition (Hourly Pricing) (preview)', 'Azure Database for MySQL', 'Azure Database for PostgreSQL', 'Azure Cosmos DB', and 'SQL Server 2017 Enterprise Windows Server 2016'.

3. Fill out the **SQL Database** form with the following information, as shown on the preceding image:

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Database name</b>	<i>yourDatabase</i>	For valid database names, see <a href="#">Database identifiers</a> .
<b>Subscription</b>	<i>yourSubscription</i>	For details about your subscriptions, see <a href="#">Subscriptions</a> .
<b>Resource group</b>	<i>yourResourceGroup</i>	For valid resource group names, see <a href="#">Naming rules and restrictions</a> .
<b>Select source</b>	Blank database	Specifies that a blank database should be created.

4. Click **Server** to use an existing database server or create and configure a new database server. Either select an existing server or click **Create a new server** and fill out the **New server** form with the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
---------	-----------------	-------------

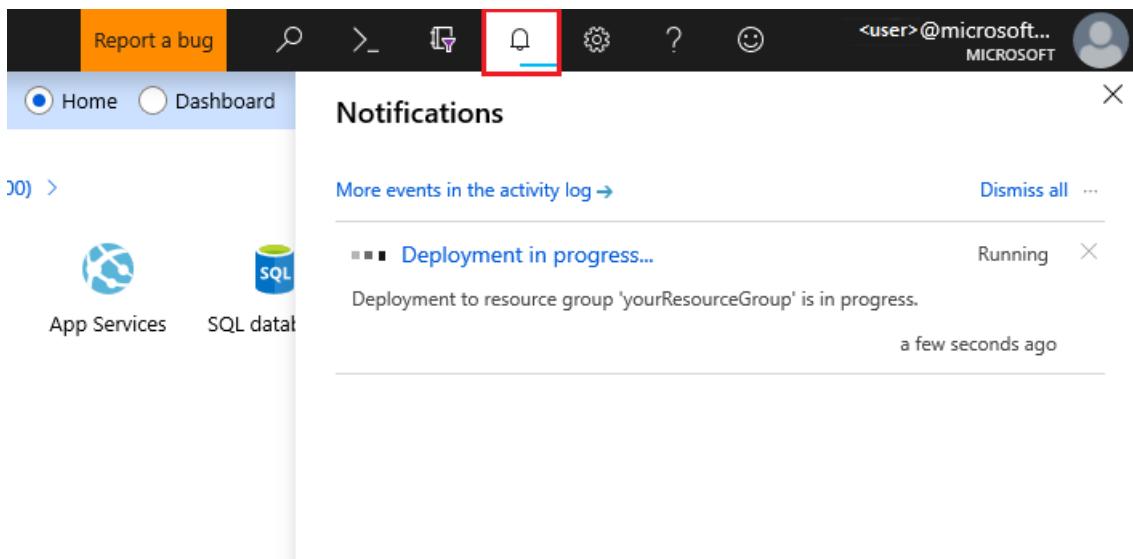
SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Server name</b>	Any globally unique name	For valid server names, see <a href="#">Naming rules and restrictions</a> .
<b>Server admin login</b>	Any valid name	For valid login names, see <a href="#">Database identifiers</a> .
<b>Password</b>	Any valid password	Your password must have at least eight characters and must use characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters.
<b>Location</b>	Any valid location	For information about regions, see <a href="#">Azure Regions</a> .

The screenshot shows the 'New server' creation dialog in the Azure portal. On the left, there's a sidebar with a 'Create a new server' button (highlighted with a red box) and a list of existing servers: 'existingserver' located in 'West US 2' with a 'yourR...' suffix. The main form contains the following fields:

- Server name:** <yourServer>.database.windows.net
- Server admin login:** <yourUser>
- Password:** (redacted)
- Confirm password:** (redacted)
- Location:** West US 2
- Allow Azure services to access server:** (checkbox checked)

A large blue 'Select' button is at the bottom right of the form.

5. Click **Select**.
  6. Click **Pricing tier** to specify the service tier, the number of DTUs or vCores, and the amount of storage. You may explore the options for the number of DTUs/vCores and storage that is available to you for each service tier.
- After selecting the service tier, the number of DTUs or vCores, and the amount of storage, click **Apply**.
7. Enter a **Collation** for the blank database (for this tutorial, use the default value). For more information about collations, see [Collations](#)
  8. Now that you've completed the **SQL Database** form, click **Create** to provision the single database. This step may take a few minutes.
  9. On the toolbar, click **Notifications** to monitor the deployment process.



## Create a server-level IP firewall rule

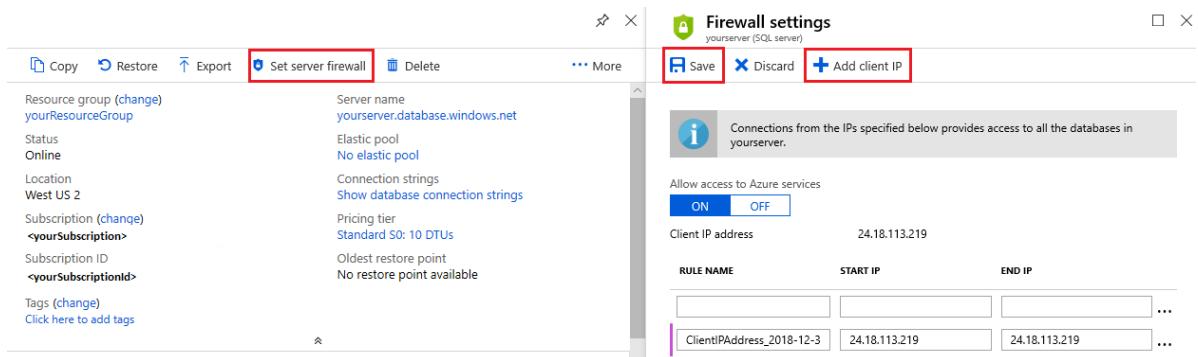
The SQL Database service creates an IP firewall at the server-level. This firewall prevents external applications and tools from connecting to the server and any databases on the server unless a firewall rule allows their IP through the firewall. To enable external connectivity to your single database, you must first add an IP firewall rule for your IP address (or IP address range). Follow these steps to create a [SQL Database server-level IP firewall rule](#).

### IMPORTANT

The SQL Database service communicates over port 1433. If you are trying to connect to this service from within a corporate network, outbound traffic over port 1433 may not be allowed by your network's firewall. If so, you cannot connect to your single database unless your administrator opens port 1433.

1. After the deployment completes, click **SQL databases** from the left-hand menu and then click *yourDatabase* on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified **Server name** (such as *yourserver.database.windows.net*) and provides options for further configuration.
2. Copy this fully qualified server name for use to connect to your server and databases from SQL Server Management Studio.

3. Click **Set server firewall** on the toolbar. The **Firewall settings** page for the SQL Database server opens.



4. Click **Add client IP** on the toolbar to add your current IP address to a new IP firewall rule. An IP firewall rule can open port 1433 for a single IP address or a range of IP addresses.
5. Click **Save**. A server-level IP firewall rule is created for your current IP address opening port 1433 on the SQL Database server.
6. Click **OK** and then close the **Firewall settings** page.

Your IP address can now pass through the IP firewall. You can now connect to your single database using SQL Server Management Studio or another tool of your choice. Be sure to use the server admin account you created previously.

#### IMPORTANT

By default, access through the SQL Database IP firewall is enabled for all Azure services. Click **OFF** on this page to disable for all Azure services.

## C# program example

The next sections of this article present a C# program that uses ADO.NET to send Transact-SQL (T-SQL) statements to the SQL database. The C# program demonstrates the following actions:

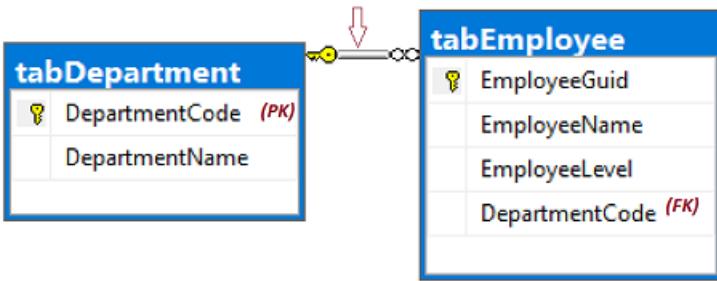
- [Connect to SQL database using ADO.NET](#)
- [Methods that return T-SQL statements](#)
  - Create tables
  - Populate tables with data
  - Update, delete, and select data
- [Submit T-SQL to the database](#)

### Entity Relationship Diagram (ERD)

The `CREATE TABLE` statements involve the **REFERENCES** keyword to create a *foreign key* (FK) relationship between two tables. If you're using `tempdb`, comment out the `--REFERENCES` keyword using a pair of leading dashes.

The ERD displays the relationship between the two tables. The values in the **tabEmployee.DepartmentCode** child column are limited to values from the **tabDepartment.DepartmentCode** parent column.

Relationship 'FK\_tabEmploy\_Depar\_412EB0B6' between 'tabDepartment' and 'tabEmployee'



#### NOTE

You have the option of editing the T-SQL to add a leading `#` to the table names, which creates them as temporary tables in `tempdb`. This is useful for demonstration purposes, when no test database is available. Any reference to foreign keys are not enforced during their use and temporary tables are deleted automatically when the connection closes after the program finishes running.

#### To compile and run

The C# program is logically one .cs file, and is physically divided into several code blocks, to make each block easier to understand. To compile and run the program, do the following steps:

1. Create a C# project in Visual Studio. The project type should be a *Console*, found under **Templates > Visual C# > Windows Desktop > Console App (.NET Framework)**.
2. In the file *Program.cs*, replace the starter lines of code with the following steps:
  - a. Copy and paste the following code blocks, in the same sequence they're presented, see [Connect to database](#), [Generate T-SQL](#), and [Submit to database](#).
  - b. Change the following values in the `Main` method:
    - `cb.DataSource`
    - `cb.UserID`
    - `cb.Password`
    - `cb.InitialCatalog`
3. Verify the assembly `System.Data.dll` is referenced. To verify, expand the **References** node in the **Solution Explorer** pane.
4. To build and run the program from Visual Studio, select the **Start** button. The report output is displayed in a program window, though GUID values will vary between test runs.

```
=====
T-SQL to 2 - Create-Tables...
-1 = rows affected.

=====
T-SQL to 3 - Inserts...
8 = rows affected.

=====
T-SQL to 4 - Update-Join...
2 = rows affected.

=====
T-SQL to 5 - Delete-Join...
2 = rows affected.

=====
Now, SelectEmployees (6)...
8ddeb8f5-9584-4afe-b7ef-d6bdca02bd35 , Alison , 20 , acct , Accounting
9ce11981-e674-42f7-928b-6cc004079b03 , Barbara , 17 , hres , Human Resources
315f5230-ec94-4edd-9b1c-dd45fb61ee7 , Carol , 22 , acct , Accounting
fcf4840a-8be3-43f7-a319-52304bf0f48d , Elle , 15 , NULL , NULL
View the report output here, then press any key to end the program...
```

## Connect to SQL database using ADO.NET

```

using System;
using System.Data.SqlClient; // System.Data.dll
//using System.Data; // For: SqlDbType , ParameterDirection

namespace csharp_db_test
{
 class Program
 {
 static void Main(string[] args)
 {
 try
 {
 var cb = new SqlConnectionStringBuilder();
 cb.DataSource = "your_server.database.windows.net";
 cb.UserID = "your_user";
 cb.Password = "your_password";
 cb.InitialCatalog = "your_database";

 using (var connection = new SqlConnection(cb.ConnectionString))
 {
 connection.Open();

 Submit_Tsql_NonQuery(connection, "2 - Create-Tables", Build_2_Tsql_CreateTables());

 Submit_Tsql_NonQuery(connection, "3 - Inserts", Build_3_Tsql_Inserts());

 Submit_Tsql_NonQuery(connection, "4 - Update-Join", Build_4_Tsql_UpdateJoin(),
 "@csharpParmDepartmentName", "Accounting");

 Submit_Tsql_NonQuery(connection, "5 - Delete-Join", Build_5_Tsql_DeleteJoin(),
 "@csharpParmDepartmentName", "Legal");

 Submit_6_Tsql_SelectEmployees(connection);
 }
 }
 catch (SqlException e)
 {
 Console.WriteLine(e.ToString());
 }

 Console.WriteLine("View the report output here, then press any key to end the program...");
 Console.ReadKey();
 }
 }
}

```

## Methods that return T-SQL statements

```

static string Build_2_Tsql_CreateTables()
{
 return @"
 DROP TABLE IF EXISTS tabEmployee;
 DROP TABLE IF EXISTS tabDepartment; -- Drop parent table last.

 CREATE TABLE tabDepartment
 (
 DepartmentCode nchar(4) not null PRIMARY KEY,
 DepartmentName nvarchar(128) not null
);

 CREATE TABLE tabEmployee
 (
 EmployeeGuid uniqueIdentifier not null default NewId() PRIMARY KEY,
 EmployeeName nvarchar(128) not null,
 EmployeeLevel int not null,
 DepartmentCode nchar(4) null
 REFERENCES tabDepartment (DepartmentCode) -- (REFERENCES would be disallowed on temporary
tables.)
 ";
}

```

```

);
 ";
}

static string Build_3_Tsql_Inserts()
{
 return @"
 -- The company has these departments.
 INSERT INTO tabDepartment (DepartmentCode, DepartmentName)
 VALUES
 ('acct', 'Accounting'),
 ('hres', 'Human Resources'),
 ('legl', 'Legal');

 -- The company has these employees, each in one department.
 INSERT INTO tabEmployee (EmployeeName, EmployeeLevel, DepartmentCode)
 VALUES
 ('Alison' , 19, 'acct'),
 ('Barbara' , 17, 'hres'),
 ('Carol' , 21, 'acct'),
 ('Deborah' , 24, 'legl'),
 ('Elle' , 15, null);
 ";
}

static string Build_4_Tsql_UpdateJoin()
{
 return @"
 DECLARE @DName1 nvarchar(128) = @csharpParmDepartmentName; --'Accounting';

 -- Promote everyone in one department (see @parm...).
 UPDATE empl
 SET
 empl.EmployeeLevel += 1
 FROM
 tabEmployee as empl
 INNER JOIN
 tabDepartment as dept ON dept.DepartmentCode = empl.DepartmentCode
 WHERE
 dept.DepartmentName = @DName1;
 ";
}

static string Build_5_Tsql_DeleteJoin()
{
 return @"
 DECLARE @DName2 nvarchar(128);
 SET @DName2 = @csharpParmDepartmentName; --'Legal';

 -- Right size the Legal department.
 DELETE empl
 FROM
 tabEmployee as empl
 INNER JOIN
 tabDepartment as dept ON dept.DepartmentCode = empl.DepartmentCode
 WHERE
 dept.DepartmentName = @DName2

 -- Disband the Legal department.
 DELETE tabDepartment
 WHERE DepartmentName = @DName2;
 ";
}

static string Build_6_Tsql_SelectEmployees()
{
 return @"
 -- Look at all the final Employees.
 SELECT

```

```
 empl.EmployeeGuid,
 empl.EmployeeName,
 empl.EmployeeLevel,
 empl.DepartmentCode,
 dept.DepartmentName
 FROM
 tabEmployee as empl
 LEFT OUTER JOIN
 tabDepartment as dept ON dept.DepartmentCode = empl.DepartmentCode
 ORDER BY
 EmployeeName;
 "
}
```

**Submit T-SQL to the database**

```

static void Submit_6_Tsql_SelectEmployees(SqlConnection connection)
{
 Console.WriteLine();
 Console.WriteLine("=====");
 Console.WriteLine("Now, SelectEmployees (6)...");

 string tsql = Build_6_Tsql_SelectEmployees();

 using (var command = new SqlCommand(tsql, connection))
 {
 using (SqlDataReader reader = command.ExecuteReader())
 {
 while (reader.Read())
 {
 Console.WriteLine("{0} , {1} , {2} , {3} , {4}",
 reader.GetGuid(0),
 reader.GetString(1),
 reader.GetInt32(2),
 (reader.IsDBNull(3)) ? "NULL" : reader.GetString(3),
 (reader.IsDBNull(4)) ? "NULL" : reader.GetString(4));
 }
 }
 }
}

static void Submit_Tsql_NonQuery(
 SqlConnection connection,
 string tsqlPurpose,
 string tsqlSourceCode,
 string parameterName = null,
 string parameterValue = null
)
{
 Console.WriteLine();
 Console.WriteLine("=====");
 Console.WriteLine("T-SQL to {0}...", tsqlPurpose);

 using (var command = new SqlCommand(tsqlSourceCode, connection))
 {
 if (parameterName != null)
 {
 command.Parameters.AddWithValue(// Or, use SqlParameter class.
 parameterName,
 parameterValue);
 }
 int rowsAffected = command.ExecuteNonQuery();
 Console.WriteLine(rowsAffected + " = rows affected.");
 }
}
} // EndOfClass
}

```

## Next steps

In this tutorial, you learned basic database tasks such as create a database and tables, connect to the database, load data, and run queries. You learned how to:

- Create a database
- Set up a firewall rule
- Connect to the database with [Visual Studio and C#](#)
- Create tables
- Insert, update, delete, and query data

Advance to the next tutorial to learn about data migration.

[Migrate SQL Server to Azure SQL Database offline using DMS](#)

# Import or export an Azure SQL database without allowing Azure services to access the server

1/10/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to import or export an Azure SQL database when *Allow Azure Services* is set to *OFF* on the Azure SQL server. The workflow uses an Azure virtual machine to run SqlPackage to perform the import or export operation.

## Sign in to the Azure portal

Sign in to the [Azure portal](#).

## Create the Azure virtual machine

Create an Azure virtual machine by selecting the **Deploy to Azure** button.

This template allows you to deploy a simple Windows virtual machine using a few different options for the Windows version, using the latest patched version. This will deploy a A2 size VM in the resource group location and return the fully qualified domain name of the VM.



For more information, see [Very simple deployment of a Windows VM](#).

## Connect to the virtual machine

The following steps show you how to connect to your virtual machine using a remote desktop connection.

1. After deployment completes, go to the virtual machine resource.

The screenshot shows the Microsoft Azure portal interface. In the center, there is a detailed view of a virtual machine named 'mymivmq'. The 'Overview' tab is selected, indicated by a red box around its title. To the right of the tabs, there are several action buttons: 'Connect' (highlighted with a red box), 'Start', 'Restart', 'Stop', 'Capture', 'Delete', and 'Refresh'. Below the tabs, the virtual machine's configuration is listed, including its resource group ('mymirg'), status ('Running'), location ('North Europe'), subscription, computer name ('mymivmq'), operating system ('Windows'), size ('Standard B2s (2 vcpus, 4 GB memory)'), public IP address, virtual network/subnet ('vnet-mymanagedinstance /Default'), and DNS name ('Configure'). On the left side, a sidebar lists various management options like Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Networking, Disks, Size, Security, Extensions, Continuous delivery (Preview), Availability set, Configuration, Identity (Preview), Properties, Locks, Automation script), and a 'Click here to add tags' button. At the bottom, there is a performance monitoring section with two charts: 'CPU (average)' and 'Network (total)'. The 'CPU (average)' chart shows usage from 0% to 100%. The 'Network (total)' chart shows traffic from 0B to 100B. Below these charts, there are time-based filters: 'Show data for last:' followed by buttons for '1 hour', '6 hours', '12 hours', '1 day', '7 days', and '30 days'.

## 2. Select **Connect**.

A Remote Desktop Protocol file (.rdp file) form appears with the public IP address and port number for the virtual machine.

The screenshot shows a modal dialog box titled 'Connect to virtual machine' for the virtual machine 'quickstartmivm'. The 'RDP' tab is selected. Inside the dialog, there are fields for 'IP address' (set to 'Public IP address (40.115.106.63)') and 'Port number' (set to '3389'). Below these fields is a large blue button labeled 'Download RDP File' with a red box highlighting it. At the bottom of the dialog, there are two informational boxes: one with an info icon stating 'Inbound traffic to the Public IP address may be blocked. You can update inbound port rules in the VM Networking page.' and another with a wrench icon stating 'You can troubleshoot VM connection issues by opening the Diagnose and solve problems page.'

## 3. Select **Download RDP File**.

#### NOTE

You can also use SSH to connect to your VM.

4. Close the **Connect to virtual machine** form.
5. To connect to your VM, open the downloaded RDP file.
6. When prompted, select **Connect**. On a Mac, you need an RDP client such as this [Remote Desktop Client](#) from the Mac App Store.
7. Enter the username and password you specified when creating the virtual machine, then choose **OK**.
8. You might receive a certificate warning during the sign-in process. Choose **Yes** or **Continue** to proceed with the connection.

## Install SqlPackage

[Download and install the latest version of SqlPackage](#).

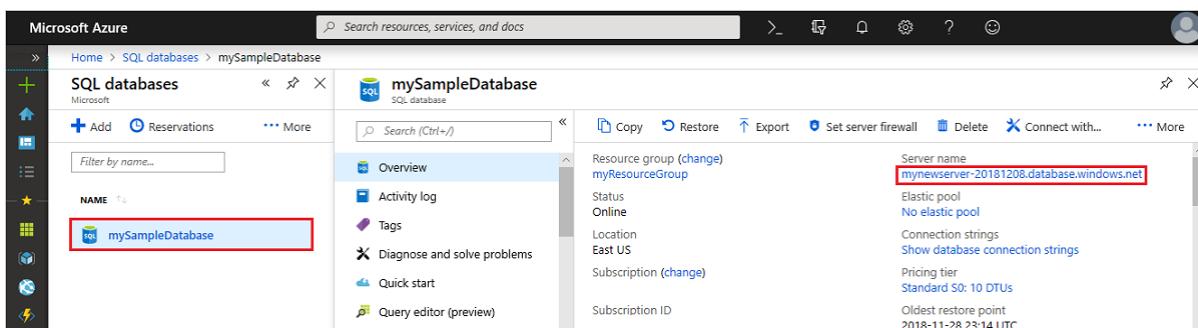
For additional information, see [SqlPackage.exe](#).

## Create a firewall rule to allow the VM access to the database

Add the virtual machine's public IP address to the SQL Database server firewall.

The following steps create a server-level IP firewall rule for your virtual machine's public IP address and enables connectivity from the virtual machine.

1. Select **SQL databases** from the left-hand menu and then select your database on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified server name (such as **servername.database.windows.net**) and provides options for further configuration.
2. Copy this fully qualified server name to use when connecting to your server and its databases.



The screenshot shows the Microsoft Azure portal interface. In the top navigation bar, 'Microsoft Azure' is at the top left, followed by a search bar 'Search resources, services, and docs'. Below the search bar, the path 'Home > SQL databases > mySampleDatabase' is visible. On the left, there's a sidebar with icons for Home, All resources, and a plus sign for adding new resources. The main content area has a title 'SQL databases' with a Microsoft logo. A sub-section for 'mySampleDatabase' is shown, with a 'mySampleDatabase' icon and the text 'SQL database'. Below this, there's a toolbar with icons for 'Add', 'Reservations', 'More', 'Copy', 'Restore', 'Export', 'Set server firewall', 'Delete', 'Connect with...', and 'More'. A 'Search (Ctrl+I)' input field is also present. The main content pane shows an 'Overview' section with a list of details: Resource group (myResourceGroup), Status (Online), Location (East US), Subscription (change), and Subscription ID. At the bottom right of the overview section, the 'Server name' is listed as 'mynewserver-20181208.database.windows.net', which is highlighted with a red rectangular box.

3. Select **Set server firewall** on the toolbar. The **Firewall settings** page for the database server opens.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various icons. The main area shows a 'Firewall settings' page for a database named 'mySampleDatabase'. The 'Set server firewall' button is at the top. Below it, there's information about the server, including its name ('mynewserver-20181208.database.windows.net'), location ('East US'), and connection strings. A 'Resource utilization' chart for 'mySampleDatabase' is shown, with a legend for '1 hour', '24 hours', and '7 days'. On the right, the 'Firewall settings' section is detailed, showing a table for 'Client IP address' with a row for 'ClientIPAddress\_2018-11' with 'START IP' as '73.42.249.7' and 'END IP' as '73.42.249.7'. There are also sections for 'Virtual networks' and 'No vnet rules for this server'.

4. Choose **Add client IP** on the toolbar to add your virtual machine's public IP address to a new server-level IP firewall rule. A server-level IP firewall rule can open port 1433 for a single IP address or a range of IP addresses.
5. Select **Save**. A server-level IP firewall rule is created for your virtual machine's public IP address opening port 1433 on the SQL Database server.
6. Close the **Firewall settings** page.

## Export a database using SqlPackage

To export a SQL database using the [SqlPackage command-line utility](#), see [Export parameters and properties](#). The SqlPackage utility ships with the latest versions of [SQL Server Management Studio](#) and [SQL Server Data Tools](#), or you can download the latest version of [SqlPackage](#).

We recommend the use of the SqlPackage utility for scale and performance in most production environments. For a SQL Server Customer Advisory Team blog about migrating using BACPAC files, see [Migrating from SQL Server to Azure SQL Database using BACPAC Files](#).

This example shows how to export a database using SqlPackage.exe with Active Directory Universal Authentication. Replace with values that are specific to your environment.

```
SqlPackage.exe /a:Export /tf:testExport.bacpac /scs:"Data Source=<servername>.database.windows.net;Initial Catalog=MyDB;" /ua:True /tid:"apptest.onmicrosoft.com"
```

## Import a database using SqlPackage

To import a SQL Server database using the [SqlPackage command-line utility](#), see [import parameters and properties](#). SqlPackage has the latest [SQL Server Management Studio](#) and [SQL Server Data Tools](#). You can also download the latest version of [SqlPackage](#).

For scale and performance, we recommend using SqlPackage in most production environments rather than using the Azure portal. For a SQL Server Customer Advisory Team blog about migrating using [BACPAC](#) files, see [migrating from SQL Server to Azure SQL Database using BACPAC Files](#).

The following SqlPackage command imports the **AdventureWorks2017** database from local storage to an Azure SQL Database server. It creates a new database called **myMigratedDatabase** with a **Premium** service tier and a **P6** Service Objective. Change these values as appropriate for your environment.

```
sqlpackage.exe /a:import /tcs:"Data Source=<serverName>.database.windows.net;Initial Catalog=myMigratedDatabase;User Id=<userId>;Password=<password>" /sf:AdventureWorks2017.bacpac /p:DatabaseEdition=Premium /p:DatabaseServiceObjective=P6
```

### IMPORTANT

To connect to a SQL Database server managing a single database from behind a corporate firewall, the firewall must have port 1433 open. To connect to a managed instance, you must have a [point-to-site connection](#) or an express route connection.

This example shows how to import a database using SqlPackage with Active Directory Universal Authentication.

```
sqlpackage.exe /a:Import /sf:testExport.bacpac /tdn>NewDacFX /tsn:apptestserver.database.windows.net /ua:True /tid:"apptest.onmicrosoft.com"
```

## Performance considerations

Export speeds vary due to many factors (for example, data shape) so it's impossible to predict what speed should be expected. SqlPackage may take considerable time, particularly for large databases.

To get the best performance you can try the following strategies:

1. Make sure no other workload is running on the database. Create a copy before export may be the best solution to ensure no other workloads are running.
2. Increase database service level objective (SLO) to better handle the export workload (primarily read I/O). If the database is currently GP\_Gen5\_4, perhaps a Business Critical tier would help with read workload.
3. Make sure there are clustered indexes particularly for large tables.
4. Virtual machines (VMs) should be in the same region as the database to help avoid network constraints.
5. VMs should have SSD with adequate size for generating temp artifacts before uploading to blob storage.
6. VMs should have adequate core and memory configuration for the specific database.

## Store the imported or exported .BACPAC file

The .BACPAC file can be stored in [Azure Blobs](#), or [Azure Files](#).

To achieve the best performance, use Azure Files. SqlPackage operates with the filesystem so it can access Azure Files directly.

To reduce cost, use Azure Blobs, which cost less than a premium Azure file share. However, it will require you to copy the [.BACPAC file](#) between the blob and the local file system before the import or export operation. As a result the process will take longer.

To upload or download .BACPAC files, see [Transfer data with AzCopy and Blob storage](#), and [Transfer data with AzCopy and file storage](#).

Depending on your environment, you might need to [Configure Azure Storage firewalls and virtual networks](#).

## Next steps

- To learn how to connect to and query an imported SQL Database, see [Quickstart: Azure SQL Database: Use SQL Server Management Studio to connect and query data](#).
- For a SQL Server Customer Advisory Team blog about migrating using BACPAC files, see [Migrating from SQL Server to Azure SQL Database using BACPAC Files](#).

- For a discussion of the entire SQL Server database migration process, including performance recommendations, see [SQL Server database migration to Azure SQL Database](#).
- To learn how to manage and share storage keys and shared access signatures securely, see [Azure Storage Security Guide](#).

# Quickstart: Import a BACPAC file to a database in Azure SQL Database

12/6/2019 • 6 minutes to read • [Edit Online](#)

You can import a SQL Server database into a database in Azure SQL Database using a [BACPAC](#) file. You can import the data from a [BACPAC](#) file stored in Azure Blob storage (standard storage only) or from local storage in an on-premises location. To maximize import speed by providing more and faster resources, scale your database to a higher service tier and compute size during the import process. You can then scale down after the import is successful.

## NOTE

The imported database's compatibility level is based on the source database's compatibility level.

## IMPORTANT

After importing your database, you can choose to operate the database at its current compatibility level (level 100 for the AdventureWorks2008R2 database) or at a higher level. For more information on the implications and options for operating a database at a specific compatibility level, see [ALTER DATABASE Compatibility Level](#). See also [ALTER DATABASE SCOPED CONFIGURATION](#) for information about additional database-level settings related to compatibility levels.

## Using Azure portal

Watch this video to see how to import from a BACPAC file in the Azure portal or continue reading below:

The [Azure portal](#) *only* supports creating a single database in Azure SQL Database and *only* from a BACPAC file stored in Azure Blob storage.

Migrating a database into a [managed instance](#) from a BACPAC file using Azure PowerShell is not currently supported. Use SQL Server Management Studio or SQLPackage instead.

## NOTE

Machines processing import/export requests submitted through the Azure portal or PowerShell need to store the BACPAC file as well as temporary files generated by the Data-Tier Application Framework (DacFX). The disk space required varies significantly among databases with the same size and can require disk space up to 3 times the size of the database. Machines running the import/export request only have 450GB local disk space. As a result, some requests may fail with the error `There is not enough space on the disk`. In this case, the workaround is to run sqlpackage.exe on a machine with enough local disk space. We encourage using SqlPackage to import/export databases larger than 150GB to avoid this issue.

1. To import from a BACPAC file into a new single database using the Azure portal, open the appropriate database server page and then, on the toolbar, select **Import database**.

The screenshot shows the Microsoft Azure portal interface for a SQL server named 'mysampleserver'. The left sidebar has a 'SQL servers' category with 'mysampleserver' selected. The main content area displays various settings for the server, including resource group ('My-LA-RG'), status ('Available'), location ('West US 2'), subscription ('Contoso, Ltd. Subscription'), and server admin ('mysampleserver'). A prominent red box highlights the 'Import database' button in the top right corner of the main content area.

2. Select the storage account and the container for the BACPAC file and then select the BACPAC file from which to import.
3. Specify the new database size (usually the same as origin) and provide the destination SQL Server credentials. For a list of possible values for a new Azure SQL database, see [Create Database](#).

The screenshot shows the 'Import database' dialog box. It includes fields for 'Subscription' (set to 'Contoso, Ltd. Subscription'), 'Database name' (set to '20190215bacpac'), 'Collation' (set to 'SQL\_Latin1\_General\_CI\_AS'), 'Authentication type' (set to 'SQL Server'), 'Server admin login' (set to 'servad'), and 'Password' (a masked input field). The 'OK' button is visible at the bottom of the dialog.

4. Click **OK**.

5. To monitor an import's progress, open the database's server page, and, under **Settings**, select **Import/Export history**. When successful, the import has a **Completed** status.

Operation	Database	Status
Export	mysampleDB	Completed
Import	20190215bacpac	Running, progress

6. To verify the database is live on the database server, select **SQL databases** and verify the new database is **Online**.

## Using SqlPackage

To import a SQL Server database using the [SqlPackage](#) command-line utility, see [import parameters and properties](#). SqlPackage has the latest [SQL Server Management Studio](#) and [SQL Server Data Tools for Visual Studio](#). You can also download the latest [SqlPackage](#) from the Microsoft download center.

For scale and performance, we recommend using SqlPackage in most production environments rather than using the Azure portal. For a SQL Server Customer Advisory Team blog about migrating using [BACPAC](#) files, see [migrating from SQL Server to Azure SQL Database using BACPAC Files](#).

The following SqlPackage command imports the **AdventureWorks2008R2** database from local storage to an Azure SQL Database server called **mynewserver20170403**. It creates a new database called **myMigratedDatabase** with a **Premium** service tier and a **P6** Service Objective. Change these values as appropriate for your environment.

```
sqlpackage.exe /a:import /tcs:"Data Source=<serverName>.database.windows.net;Initial Catalog=<migratedDatabase>;User Id=<userId>;Password=<password>" /sf:AdventureWorks2008R2.bacpac /p:DatabaseEdition=Premium /p:DatabaseServiceObjective=P6
```

### IMPORTANT

To connect to a SQL Database server managing a single database from behind a corporate firewall, the firewall must have port 1433 open. To connect to a managed instance, you must have a [point-to-site connection](#) or an express route connection.

This example shows how to import a database using SqlPackage with Active Directory Universal

Authentication.

```
sqlpackage.exe /a:Import /sf:testExport.bacpac /tdn>NewDacFX /tsn:apptestserver.database.windows.net
/ua:True /tid:"apptest.onmicrosoft.com"
```

## Using PowerShell

### NOTE

A managed instance does not currently support migrating a database into an instance database from a BACPAC file using Azure PowerShell. To import into a managed instance, use SQL Server Management Studio or SQLPackage.

### NOTE

The machines processing import/export requests submitted through portal or Powershell need to store the bacpac file as well as temporary files generated by Data-Tier Application Framework (DacFX). The disk space required varies significantly among DBs with same size and can take up to 3 times of the database size. Machines running the import/export request only have 450GB local disk space. As result, some requests may fail with "There is not enough space on the disk" error. In this case, the workaround is to run sqlpackage.exe on a machine with enough local disk space. When importing/exporting databases larger than 150GB, use SqlPackage to avoid this issue.

- [PowerShell](#)
- [Azure CLI](#)

### IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

Use the [New-AzSqlDatabaseImport](#) cmdlet to submit an import database request to the Azure SQL Database service. Depending on database size, the import may take some time to complete.

```
$importRequest = New-AzSqlDatabaseImport -ResourceGroupName "<resourceGroupName>" `
-ServerName "<serverName>" -DatabaseName "<databaseName>" `
-DatabaseMaxSizeBytes "<databaseSizeInBytes>" -StorageKeyType "StorageAccessKey" `
-StorageKey $(Get-AzStorageAccountKey `
-ResourceGroupName "<resourceGroupName>" -StorageAccountName "<storageAccountName>").Value[0] `
-StorageUri "https://myStorageAccount.blob.core.windows.net/importsample/sample.bacpac" `
-Edition "Standard" -ServiceObjectiveName "P6" `
-AdministratorLogin "<userId>" `
-AdministratorLoginPassword $(ConvertTo-SecureString -String "<password>" -AsPlainText -Force)
```

You can use the [Get-AzSqlDatabaseImportExportStatus](#) cmdlet to check the import's progress. Running the cmdlet immediately after the request usually returns `Status: InProgress`. The import is complete when you see `Status: Succeeded`.

```
$importStatus = Get-AzSqlDatabaseImportExportStatus -OperationStatusLink $importRequest.OperationStatusLink

[Console]::Write("Importing")
while ($importStatus.Status -eq "InProgress") {
 $importStatus = Get-AzSqlDatabaseImportExportStatus -OperationStatusLink
 $importRequest.OperationStatusLink
 [Console]::Write(".")
 Start-Sleep -s 10
}

[Console]::WriteLine("")
$importStatus
```

#### TIP

For another script example, see [Import a database from a BACPAC file](#).

## Limitations

Importing to a database in elastic pool isn't supported. You can import data into a single database and then move the database to an elastic pool.

## Import using wizards

You can also use these wizards.

- [Import Data-tier Application Wizard in SQL Server Management Studio](#).
- [SQL Server Import and Export Wizard](#).

## Next steps

- To learn how to connect to and query an imported SQL Database, see [Quickstart: Azure SQL Database: Use SQL Server Management Studio to connect and query data](#).
- For a SQL Server Customer Advisory Team blog about migrating using BACPAC files, see [Migrating from SQL Server to Azure SQL Database using BACPAC Files](#).
- For a discussion of the entire SQL Server database migration process, including performance recommendations, see [SQL Server database migration to Azure SQL Database](#).
- To learn how to manage and share storage keys and shared access signatures securely, see [Azure Storage Security Guide](#).

# Export an Azure SQL database to a BACPAC file

11/7/2019 • 5 minutes to read • [Edit Online](#)

When you need to export a database for archiving or for moving to another platform, you can export the database schema and data to a [BACPAC](#) file. A BACPAC file is a ZIP file with an extension of BACPAC containing the metadata and data from a SQL Server database. A BACPAC file can be stored in Azure Blob storage or in local storage in an on-premises location and later imported back into Azure SQL Database or into a SQL Server on-premises installation.

## Considerations when exporting an Azure SQL database

- For an export to be transactionally consistent, you must ensure either that no write activity is occurring during the export, or that you are exporting from a [transactionally consistent copy](#) of your Azure SQL database.
- If you are exporting to blob storage, the maximum size of a BACPAC file is 200 GB. To archive a larger BACPAC file, export to local storage.
- Exporting a BACPAC file to Azure premium storage using the methods discussed in this article is not supported.
- Storage behind a firewall is currently not supported.
- If the export operation from Azure SQL Database exceeds 20 hours, it may be canceled. To increase performance during export, you can:
  - Temporarily increase your compute size.
  - Cease all read and write activity during the export.
  - Use a [clustered index](#) with non-null values on all large tables. Without clustered indexes, an export may fail if it takes longer than 6-12 hours. This is because the export service needs to complete a table scan to try to export entire table. A good way to determine if your tables are optimized for export is to run **DBCC SHOW\_STATISTICS** and make sure that the *RANGE\_HI\_KEY* is not null and its value has good distribution. For details, see [DBCC SHOW\\_STATISTICS](#).

### NOTE

BACPACs are not intended to be used for backup and restore operations. Azure SQL Database automatically creates backups for every user database. For details, see [business continuity overview](#) and [SQL Database backups](#).

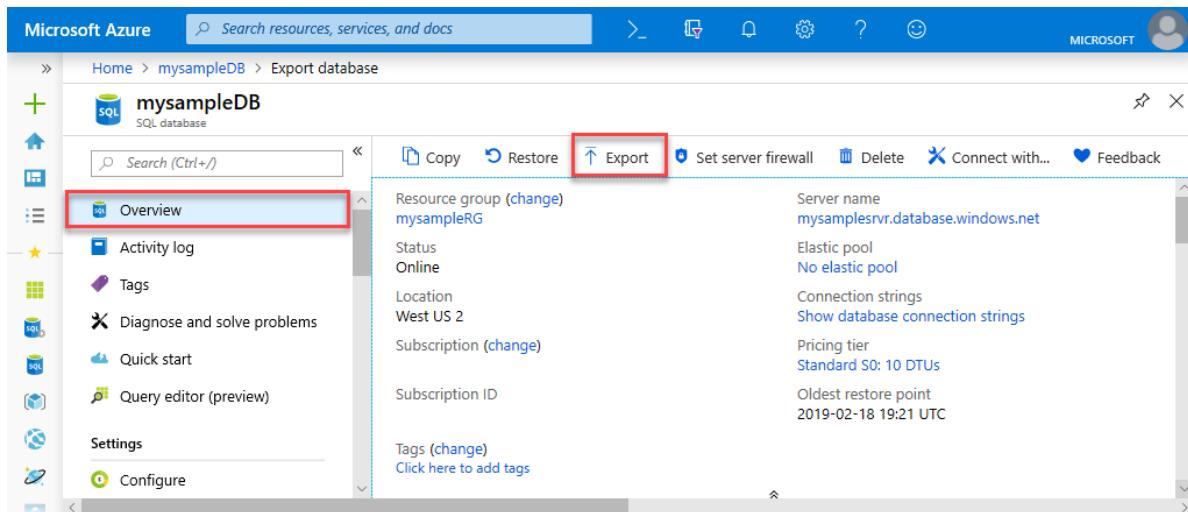
## Export to a BACPAC file using the Azure portal

Exporting a BACPAC of a database from a [managed instance](#) using Azure PowerShell is not currently supported. Use SQL Server Management Studio or SQLPackage instead.

#### NOTE

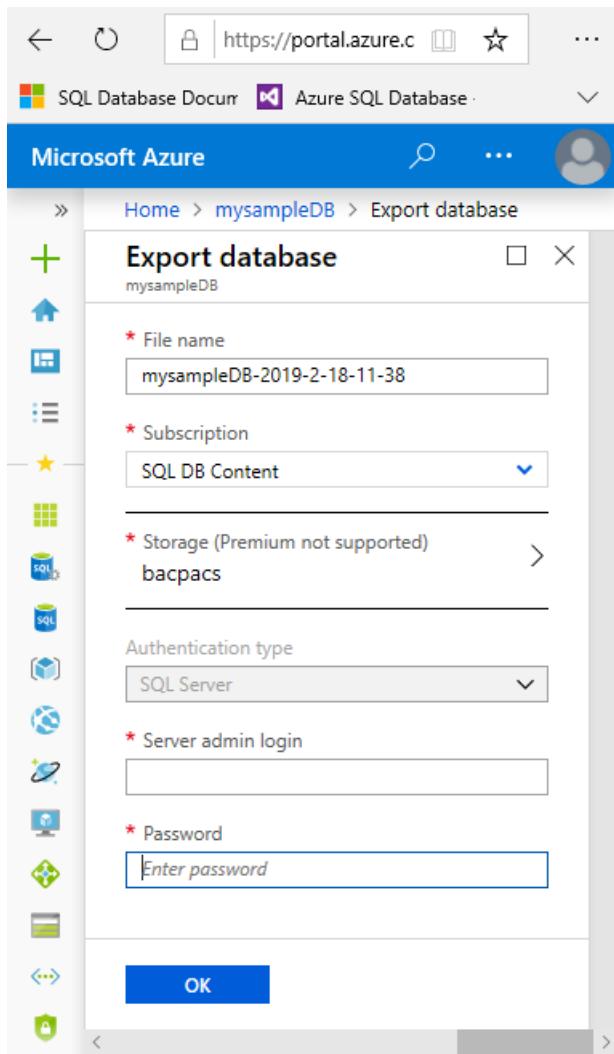
Machines processing import/export requests submitted through the Azure portal or PowerShell need to store the BACPAC file as well as temporary files generated by the Data-Tier Application Framework (DacFX). The disk space required varies significantly among databases with the same size and can require disk space up to 3 times the size of the database. Machines running the import/export request only have 450GB local disk space. As a result, some requests may fail with the error `There is not enough space on the disk`. In this case, the workaround is to run `sqlpackage.exe` on a machine with enough local disk space. We encourage using [SqlPackage](#) to import/export databases larger than 150GB to avoid this issue.

1. To export a database using the [Azure portal](#), open the page for your database and click **Export** on the toolbar.



The screenshot shows the Microsoft Azure portal interface for managing a database named 'mysampleDB'. The left sidebar contains navigation links like Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Settings, and Configure. The main content area displays database details such as Resource group (mysampleRG), Status (Online), Location (West US 2), Subscription (change), Subscription ID, and Tags (change). At the top right, there are several action buttons: Copy, Restore, Export (which is highlighted with a red box), Set server firewall, Delete, Connect with..., and Feedback.

2. Specify the BACPAC filename, select an existing Azure storage account and container for the export, and then provide the appropriate credentials for access to the source database. A SQL **Server admin login** is needed here even if you are the Azure admin, as being an Azure admin does not equate to having SQL Server admin permissions.



3. Click **OK**.
4. To monitor the progress of the export operation, open the page for the SQL Database server containing the database being exported. Under to **Settings** and then click **Import/Export history**.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar has the URL https://portal.azure.com. The main content area is titled 'Microsoft Azure' with a breadcrumb trail: Home > mysampleDB > mysamplesrvr - Import/Export history. The sidebar on the left lists various server settings: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with sub-options like Quick start, Failover groups, Manage Backups, Active Directory admin, SQL databases, SQL elastic pools, Deleted databases), and Import/Export history. The 'Import/Export history' option is highlighted with a red box. The right pane displays a table titled 'Import/Export history' with the following data:

OPERATION	DATABASE	STATUS	COMPLETION TIME
Export	mysampleDB	Running, Progress = ...	2/18/2019 7:42:23 PM

# Export to a BACPAC file using the SQLPackage utility

To export a SQL database using the [SqlPackage](#) command-line utility, see [Export parameters and properties](#). The SQLPackage utility ships with the latest versions of [SQL Server Management Studio](#) and [SQL Server Data Tools for Visual Studio](#), or you can download the latest version of [SqlPackage](#) directly from the Microsoft download center.

We recommend the use of the SQLPackage utility for scale and performance in most production environments. For a SQL Server Customer Advisory Team blog about migrating using BACPAC files, see [Migrating from SQL Server to Azure SQL Database using BACPAC Files](#).

This example shows how to export a database using SqlPackage.exe with Active Directory Universal Authentication:

```
SqlPackage.exe /a:Export /tf:testExport.bacpac /scs:"Data Source=apptestserver.database.windows.net;Initial Catalog=MyDB;" /ua:True /tid:"apptest.onmicrosoft.com"
```

## Export to a BACPAC file using SQL Server Management Studio (SSMS)

The newest versions of SQL Server Management Studio provides a wizard to export an Azure SQL database to a BACPAC file. See the [Export a Data-tier Application](#).

## Export to a BACPAC file using PowerShell

### NOTE

A [managed instance](#) does not currently support exporting a database to a BACPAC file using Azure PowerShell. To export a managed instance into a BACPAC file, use SQL Server Management Studio or SQLPackage.

Use the [New-AzSqlDatabaseExport](#) cmdlet to submit an export database request to the Azure SQL Database service. Depending on the size of your database, the export operation may take some time to complete.

```
$exportRequest = New-AzSqlDatabaseExport -ResourceGroupName $ResourceGroupName -ServerName $ServerName ` -DatabaseName $DatabaseName -StorageKeytype $StorageKeytype -StorageKey $StorageKey -StorageUri $BacpacUri -AdministratorLogin $creds.UserName -AdministratorLoginPassword $creds.Password
```

To check the status of the export request, use the [Get-AzSqlDatabaseImportExportStatus](#) cmdlet. Running this immediately after the request usually returns **Status: InProgress**. When you see **Status: Succeeded** the export is complete.

```
$exportStatus = Get-AzSqlDatabaseImportExportStatus -OperationStatusLink $exportRequest.OperationStatusLink [Console]::Write("Exporting") while ($exportStatus.Status -eq "InProgress") { Start-Sleep -s 10 $exportStatus = Get-AzSqlDatabaseImportExportStatus -OperationStatusLink $exportRequest.OperationStatusLink [Console]::Write(".") } [Console]::WriteLine("") $exportStatus
```

## Next steps

- To learn about long-term backup retention of a single databases and pooled databases as an alternative to exported a database for archive purposes, see [Long-term backup retention](#). You can use SQL Agent jobs to schedule [copy-only database backups](#) as an alternative to long-term backup retention.
- For a SQL Server Customer Advisory Team blog about migrating using BACPAC files, see [Migrating from SQL Server to Azure SQL Database using BACPAC Files](#).
- To learn about importing a BACPAC to a SQL Server database, see [Import a BACPAC to a SQL Server database](#).
- To learn about exporting a BACPAC from a SQL Server database, see [Export a Data-tier Application](#)
- To learn about using the Data Migration Service to migrate a database, see [Migrate SQL Server to Azure SQL Database offline using DMS](#).
- If you are exporting from SQL Server as a prelude to migration to Azure SQL Database, see [Migrate a SQL Server database to Azure SQL Database](#).
- To learn how to manage and share storage keys and shared access signatures securely, see [Azure Storage Security Guide](#).

# Copy a transactionally consistent copy of an Azure SQL database

2/25/2020 • 9 minutes to read • [Edit Online](#)

Azure SQL Database provides several methods for creating a transactionally consistent copy of an existing Azure SQL database ([single database](#)) on either the same server or a different server. You can copy a SQL database by using the Azure portal, PowerShell, or T-SQL.

## Overview

A database copy is a snapshot of the source database as of the time of the copy request. You can select the same server or a different server. Also you can choose to keep its service tier and compute size, or use a different compute size within the same service tier (edition). After the copy is complete, it becomes a fully functional, independent database. At this point, you can upgrade or downgrade it to any edition. The logins, users, and permissions can be managed independently. The copy is created using the geo-replication technology and once seeding is completed the geo-replication link is automatically terminated. All the requirements for using geo-replication apply to the database copy operation. See [Active geo-replication overview](#) for details.

**NOTE**

[Automated database backups](#) are used when you create a database copy.

## Logins in the database copy

When you copy a database to the same SQL Database server, the same logins can be used on both databases. The security principal you use to copy the database becomes the database owner on the new database.

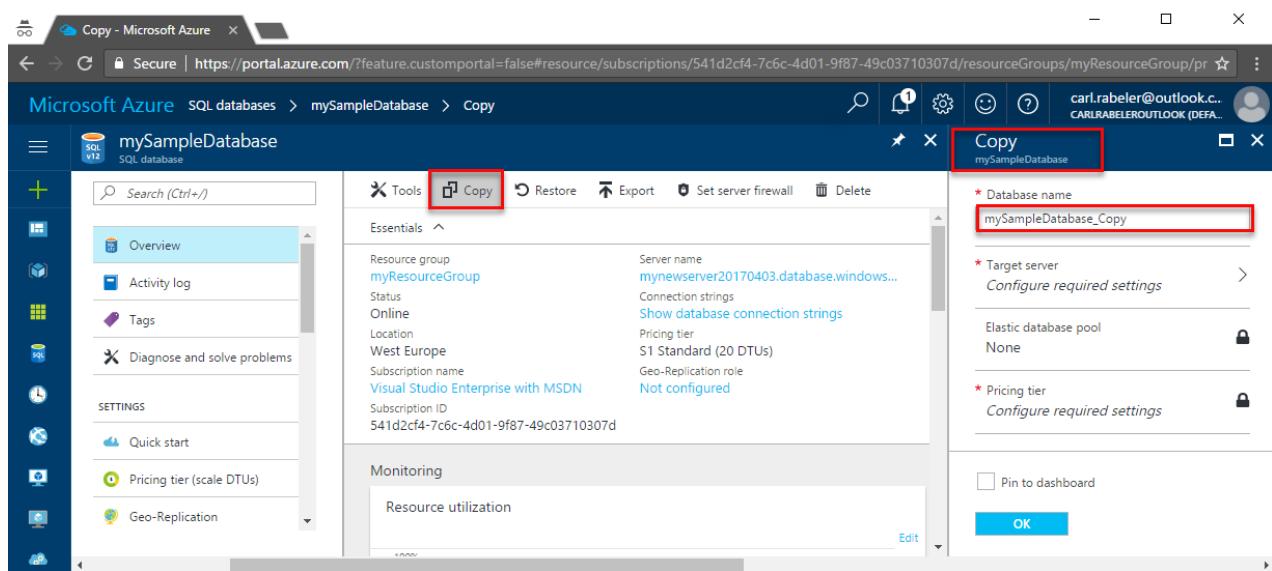
When you copy a database to a different SQL Database server, the security principal that initiated the copy operation on the target server becomes the owner of the new database.

Regardless of the target server, all database users, their permissions, and their security identifiers (SIDs) are copied to the database copy. Using [contained database users](#) for data access ensures that the copied database has the same user credentials, so that after the copy is complete you can immediately access it with the same credentials.

If you use server level logins for data access and copy the database to a different server, the login-based access might not work. This can happen because the logins do not exist on the target server, or because their passwords and security identifiers (SIDs) are different. To learn about managing logins when you copy a database to a different SQL Database server, see [How to manage Azure SQL database security after disaster recovery](#). After the copy operation to a different server succeeds, and before other users are remapped, only the login associated with the database owner, or the server administrator can log in to the copied database. To resolve logins and establish data access after the copying operation is complete, see [Resolve logins](#).

## Copy a database by using the Azure portal

To copy a database by using the Azure portal, open the page for your database, and then click **Copy**.



## Copy a database by using PowerShell or Azure CLI

To copy a database, use the following examples.

- [PowerShell](#)
- [Azure CLI](#)

For PowerShell, use the [New-AzSqlDatabaseCopy](#) cmdlet.

### IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

```
New-AzSqlDatabaseCopy -ResourceGroupName "<resourceGroup>" -ServerName $sourceserver -DatabaseName "<databaseName>"
-CopyResourceGroupName "myResourceGroup" -CopyServerName $targetserver -CopyDatabaseName
"CopyOfMySampleDatabase"
```

The database copy is an asynchronous operation but the target database is created immediately after the request is accepted. If you need to cancel the copy operation while still in progress, drop the target database using the [Remove-AzSqlDatabase](#) cmdlet.

For a complete sample PowerShell script, see [Copy a database to a new server](#).

## RBAC roles to manage database copy

To create a database copy, you will need to be in the following roles

- Subscription Owner or
- SQL Server Contributor role or
- Custom role on the source and target databases with following permission:

Microsoft.Sql/servers/databases/read Microsoft.Sql/servers/databases/write

To cancel a database copy, you will need to be in the following roles

- Subscription Owner or
- SQL Server Contributor role or
- Custom role on the source and target databases with following permission:

Microsoft.Sql/servers/databases/read Microsoft.Sql/servers/databases/write

To manage database copy using Azure portal, you will also need the following permissions:

Microsoft.Resources/subscriptions/resources/read Microsoft.Resources/subscriptions/resources/write

Microsoft.Resources/deployments/read Microsoft.Resources/deployments/write

Microsoft.Resources/deployments/operationstatuses/read

If you want to see the operations under deployments in the resource group on the portal, operations across multiple resource providers including SQL operations, you will need these additional RBAC roles:

Microsoft.Resources/subscriptions/resourcegroups/deployments/operations/read

Microsoft.Resources/subscriptions/resourcegroups/deployments/operationstatuses/read

## Copy a database by using Transact-SQL

Log in to the master database with the server administrator login or the login that created the database you want to copy. For database copy to succeed, logins that are not the server administrator must be members of the `dbmanager` role. For more information about logins and connecting to the server, see [Manage logins](#).

Start copying the source database with the `CREATE DATABASE ... AS COPY OF` statement. The T-SQL statement continues running until the database copy operation is complete.

### NOTE

Terminating the T-SQL statement does not terminate the database copy operation. To terminate the operation, drop the target database.

### Copy a SQL database to the same server

Log in to the master database with the server administrator login or the login that created the database you want to copy. For database copying to succeed, logins that are not the server administrator must be members of the `dbmanager` role.

This command copies Database1 to a new database named Database2 on the same server. Depending on the size of your database, the copying operation might take some time to complete.

```
-- execute on the master database to start copying
CREATE DATABASE Database2 AS COPY OF Database1;
```

### Copy a SQL database to a different server

Log in to the master database of the target server where the new database is to be created. Use a login that has the same name and password as the database owner of the source database on the source server. The login on the target server must also be a member of the `dbmanager` role, or be the server administrator login.

This command copies Database1 on server1 to a new database named Database2 on server2. Depending on the size of your database, the copying operation might take some time to complete.

```
-- Execute on the master database of the target server (server2) to start copying from Server1 to Server2
CREATE DATABASE Database2 AS COPY OF server1.Database1;
```

## IMPORTANT

Both servers' firewalls must be configured to allow inbound connection from the IP of the client issuing the T-SQL CREATE DATABASE ... AS COPY OF command.

## Copy a SQL database to a different subscription

You can use the steps in the [Copy a SQL database to a different server](#) section to copy your database to a SQL Database server in a different subscription using T-SQL. Make sure you use a login that has the same name and password as the database owner of the source database. Additionally, the login must be a member of the `dbmanager` role or a server administrator, on both source and target servers.

## NOTE

The [Azure portal](#), PowerShell, and Azure CLI do not support database copy to a different subscription.

## Monitor the progress of the copying operation

Monitor the copying process by querying the `sys.databases`, `sys.dm_database_copies`, and `sys.dm_operation_status` views. While the copying is in progress, the `state_desc` column of the `sys.databases` view for the new database is set to **COPYING**.

- If the copying fails, the `state_desc` column of the `sys.databases` view for the new database is set to **SUSPECT**. Execute the `DROP` statement on the new database, and try again later.
- If the copying succeeds, the `state_desc` column of the `sys.databases` view for the new database is set to **ONLINE**. The copying is complete, and the new database is a regular database that can be changed independent of the source database.

## NOTE

If you decide to cancel the copying while it is in progress, execute the `DROP DATABASE` statement on the new database.

## IMPORTANT

If you need to create a copy with a substantially smaller service objective than the source, the target database may not have sufficient resources to complete the seeding process and it can cause the copy operation to fail. In this scenario use a geo-restore request to create a copy in a different server and/or a different region. See [Recover an Azure SQL database using database backups](#) for more information.

## Resolve logins

After the new database is online on the target server, use the `ALTER USER` statement to remap the users from the new database to logins on the target server. To resolve orphaned users, see [Troubleshoot Orphaned Users](#). See also [How to manage Azure SQL database security after disaster recovery](#).

All users in the new database retain the permissions that they had in the source database. The user who initiated the database copy becomes the database owner of the new database. After the copying succeeds and before other users are remapped, only the database owner can log in to the new database.

To learn about managing users and logins when you copy a database to a different SQL Database server, see [How to manage Azure SQL database security after disaster recovery](#).

## Database copy errors

The following errors can be encountered while copying a database in Azure SQL Database. For more information, see [Copy an Azure SQL Database](#).

ERROR CODE	SEVERITY	DESCRIPTION
40635	16	Client with IP address '%.*ls' is temporarily disabled.
40637	16	Create database copy is currently disabled.
40561	16	Database copy failed. Either the source or target database does not exist.
40562	16	Database copy failed. The source database has been dropped.
40563	16	Database copy failed. The target database has been dropped.
40564	16	Database copy failed due to an internal error. Please drop target database and try again.
40565	16	Database copy failed. No more than 1 concurrent database copy from the same source is allowed. Please drop target database and try again later.
40566	16	Database copy failed due to an internal error. Please drop target database and try again.
40567	16	Database copy failed due to an internal error. Please drop target database and try again.
40568	16	Database copy failed. Source database has become unavailable. Please drop target database and try again.
40569	16	Database copy failed. Target database has become unavailable. Please drop target database and try again.
40570	16	Database copy failed due to an internal error. Please drop target database and try again later.
40571	16	Database copy failed due to an internal error. Please drop target database and try again later.

## Next steps

- For information about logins, see [Manage logins](#) and [How to manage Azure SQL database security after disaster recovery](#).

- To export a database, see [Export the database to a BACPAC](#).

# How to move Azure SQL resources to another region

11/7/2019 • 9 minutes to read • [Edit Online](#)

This article teaches you a generic workflow for how to move your Azure SQL Database single database, elastic pool, and managed instance to a new region.

## Overview

There are various scenarios in which you'd want to move your existing Azure SQL resources from one region to another. For example, you expand your business to a new region and want to optimize it for the new customer base. Or you need to move the operations to a different region for compliance reasons. Or Azure released a brand-new region that provides a better proximity and improves the customer experience.

This article provides a general workflow for moving resources to a different region. The workflow consists of the following steps:

- Verify the prerequisites for the move
- Prepare to move the resources in scope
- Monitor the preparation process
- Test the move process
- Initiate the actual move
- Remove the resources from the source region

### NOTE

This article applies to migrations within the Azure public cloud, or within the same sovereign cloud.

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## Move single database

### Verify prerequisites

1. Create a target logical server for each source server.
2. Configure the firewall with the right exceptions using [PowerShell](#).
3. Configure the logical servers with the correct logins. If you're not the subscription administrator or SQL server administrator, work with the administrator to assign the permissions that you need. For more information, see [How to manage Azure SQL database security after disaster recovery](#).
4. If your databases are encrypted with TDE and use your own encryption key in Azure key vault, ensure that the correct encryption material is provisioned in the target regions. For more information, see [Azure SQL Transparent Data Encryption with customer-managed keys in Azure Key Vault](#)
5. If database-level audit is enabled, disable it and enable server-level auditing instead. After failover, database level auditing will require the cross-region traffic, which will is not desired or possible after the move.

6. For server-level audits, ensure that:
  - The storage container, Log Analytics, or event hub with the existing audit logs is moved to the target region.
  - Auditing is configured on the target server. For more information, see [Get started with SQL database auditing](#).
7. If your instance has a long-term retention policy (LTR), the existing LTR backups will remain associated with the current server. Because the target server is different, you will be able to access the older LTR backups in the source region using the source server, even if the server is deleted.

#### NOTE

This will be insufficient for moving between the sovereign cloud and a public region. Such a migration will require moving the LTR backups to the target server, which is not currently supported.

## Prepare resources

1. Create a [failover group](#) between the logical server of the source to the logical server of the target.
2. Add the databases you want to move to the failover group.
  - Replication of all added databases will be initiated automatically. For more information, see [Best practices for using failover groups with single databases](#).

## Monitor the preparation process

You can periodically call [Get-AzSqlDatabaseFailoverGroup](#) to monitor replication of your databases from the source to the target. The output object of [Get-AzSqlDatabaseFailoverGroup](#) includes a property for the **ReplicationState**:

- **ReplicationState = 2** (CATCH\_UP) indicates the database is synchronized and can be safely failed over.
- **ReplicationState = 0** (SEEDING) indicates that the database is not yet seeded, and an attempt to failover will fail.

## Test synchronization

Once **ReplicationState** is **2**, connect to each database, or subset of databases using the secondary endpoint `<fog-name>.secondary.database.windows.net` and perform any query against the databases to ensure connectivity, proper security configuration, and data replication.

## Initiate the move

1. Connect to the target server using the secondary endpoint `<fog-name>.secondary.database.windows.net`.
2. Use [Switch-AzSqlDatabaseFailoverGroup](#) to switch the secondary managed instance to be the primary with full synchronization. This operation will either succeed, or it will roll back.
3. Verify that the command has completed successfully by using `nslookup <fog-name>.secondary.database.windows.net` to ascertain that the DNS CNAME entry points to the target region IP address. If the switch command fails, the CNAME will not get updated.

## Remove the source databases

Once the move completes, remove the resources in the source region to avoid unnecessary charges.

1. Delete the failover group using [Remove-AzSqlDatabaseFailoverGroup](#).
2. Delete each source database using [Remove-AzSqlDatabase](#) for each of the databases on the source server. This will automatically terminate geo-replication links.
3. Delete the source server using [Remove-AzSqlServer](#).
4. Remove the key vault, audit storage containers, event hub, AAD instance, and other dependent resources to stop being billed for them.

# Move elastic pools

## Verify prerequisites

1. Create a target logical server for each source server.
2. Configure the firewall with the right exceptions using [PowerShell](#).
3. Configure the logical servers with the correct logins. If you're not the subscription administrator or SQL server administrator, work with the administrator to assign the permissions that you need. For more information, see [How to manage Azure SQL database security after disaster recovery](#).
4. If your databases are encrypted with TDE and use your own encryption key in Azure key vault, ensure that the correct encryption material is provisioned in the target region.
5. Create a target elastic pool for each source elastic pool, making sure the pool is created in the same service tier, with the same name and the same size.
6. If a database-level audit is enabled, disable it and enable server-level auditing instead. After failover, database-level auditing will require cross-region traffic, which is not desired, or possible after the move.
7. For server-level audits, ensure that:
  - The storage container, Log Analytics, or event hub with the existing audit logs is moved to the target region.
  - Audit configuration is configured at the target server. For more information, see [SQL database auditing](#).
8. If your instance has a long-term retention policy (LTR), the existing LTR backups will remain associated with the current server. Because the target server is different, you will be able to access the older LTR backups in the source region using the source server, even if the server is deleted.

### NOTE

This will be insufficient for moving between the sovereign cloud and a public region. Such a migration will require moving the LTR backups to the target server, which is not currently supported.

## Prepare to move

1. Create a separate [failover group](#) between each elastic pool on the source logical server and its counterpart elastic pool on the target server.
2. Add all the databases in the pool to the failover group.
  - Replication of the added databases will be initiated automatically. For more information, see [best practices for failover groups with elastic pools](#).

### NOTE

While it is possible to create a failover group that includes multiple elastic pools, we strongly recommend that you create a separate failover group for each pool. If you have a large number of databases across multiple elastic pools that you need to move, you can run the preparation steps in parallel and then initiate the move step in parallel. This process will scale better and will take less time compared to having multiple elastic pools in the same failover group.

## Monitor the preparation process

You can periodically call `Get-AzSqlDatabaseFailoverGroup` to monitor replication of your databases from the source to the target. The output object of `Get-AzSqlDatabaseFailoverGroup` includes a property for the **ReplicationState**:

- **ReplicationState = 2** (CATCH\_UP) indicates the database is synchronized and can be safely failed over.
- **ReplicationState = 0** (SEEDING) indicates that the database is not yet seeded, and an attempt to failover will fail.

## Test synchronization

Once **ReplicationState** is `2`, connect to each database, or subset of databases using the secondary endpoint `<fog-name>.secondary.database.windows.net` and perform any query against the databases to ensure connectivity, proper security configuration, and data replication.

## Initiate the move

1. Connect to the target server using the secondary endpoint `<fog-name>.secondary.database.windows.net`.
2. Use [Switch-AzSqlDatabaseFailoverGroup](#) to switch the secondary managed instance to be the primary with full synchronization. This operation will either succeed, or it will roll back.
3. Verify that the command has completed successfully by using `nslookup <fog-name>.secondary.database.windows.net` to ascertain that the DNS CNAME entry points to the target region IP address. If the switch command fails, the CNAME will not get updated.

## Remove the source elastic pools

Once the move completes, remove the resources in the source region to avoid unnecessary charges.

1. Delete the failover group using [Remove-AzSqlDatabaseFailoverGroup](#).
2. Delete each source elastic pool on the source server using [Remove-AzSqlElasticPool](#).
3. Delete the source server using [Remove-AzSqlServer](#).
4. Remove the key vault, audit storage containers, event hub, AAD instance, and other dependent resources to stop being billed for them.

# Move managed instance

## Verify prerequisites

1. For each source managed instance create a target managed instance of the same size in the target region.
2. Configure the network for a managed instance. For more information, see [network configuration](#).
3. Configure the target master database with the correct logins. If you're not the subscription administrator or SQL server administrator, work with the administrator to assign the permissions that you need.
4. If your databases are encrypted with TDE and use your own encryption key in Azure key vault, ensure that the AKV with identical encryption keys exists in both source and target regions. For more information, see [TDE with customer-managed keys in Azure Key Vault](#).
5. If audit is enabled for the instance, ensure that:
  - The storage container or event hub with the existing logs is moved to the target region.
  - Audit is configured on the target instance. For more information, see [auditing with managed instance](#).
6. If your instance has a long-term retention policy (LTR), the existing LTR backups will remain associated with the current server. Because the target server is different, you will be able to access the older LTR backups in the source region using the source server, even if the server is deleted.

### NOTE

This will be insufficient for moving between the sovereign cloud and a public region. Such a migration will require moving the LTR backups to the target server, which is not currently supported.

## Prepare resources

Create a failover group between each source instance and the corresponding target instance. - Replication of all databases on each instance will be initiated automatically. See [Auto-failover groups](#) for more information.

## Monitor the preparation process

You can periodically call [Get-AzSqlDatabaseFailoverGroup](#) to monitor replication of your databases from the source to the target. The output object of `Get-AzSqlDatabaseFailoverGroup` includes a property for the **ReplicationState**:

- **ReplicationState = 2** (CATCH\_UP) indicates the database is synchronized and can be safely failed over.
- **ReplicationState = 0** (SEEDING) indicates that the database is not yet seeded, and an attempt to failover will fail.

## Test synchronization

Once **ReplicationState** is **2**, connect to each database, or subset of databases using the secondary endpoint `<fog-name>.secondary.database.windows.net` and perform any query against the databases to ensure connectivity, proper security configuration, and data replication.

## Initiate the move

1. Connect to the target server using the secondary endpoint `<fog-name>.secondary.database.windows.net`.
2. Use [Switch-AzSqlDatabaseFailoverGroup](#) to switch the secondary managed instance to be the primary with full synchronization. This operation will either succeed, or it will roll back.
3. Verify that the command has completed successfully by using  
`nslookup <fog-name>.secondary.database.windows.net` to ascertain that the DNS CNAME entry points to the target region IP address. If the switch command fails, the CNAME will not get updated.

## Remove the source managed instances

Once the move completes, remove the resources in the source region to avoid unnecessary charges.

1. Delete the failover group using [Remove-AzSqlDatabaseFailoverGroup](#). This will drop the failover group configuration and terminate geo-replication links between the two instances.
2. Delete the source managed instance using [Remove-AzSqlInstance](#).
3. Remove any additional resources in the resource group, such as the virtual cluster, virtual network, and security group.

## Next steps

[Manage](#) your Azure SQL Database once it's been migrated.

# Load data from CSV into Azure SQL Database (flat files)

11/7/2019 • 2 minutes to read • [Edit Online](#)

You can use the bcp command-line utility to import data from a CSV file into Azure SQL Database.

## Before you begin

### Prerequisites

To complete the steps in this article, you need:

- An Azure SQL Database server and database
- The bcp command-line utility installed
- The sqlcmd command-line utility installed

You can download the bcp and sqlcmd utilities from the [Microsoft Download Center](#).

### Data in ASCII or UTF-16 format

If you are trying this tutorial with your own data, your data needs to use the ASCII or UTF-16 encoding since bcp does not support UTF-8.

## 1. Create a destination table

Define a table in SQL Database as the destination table. The columns in the table must correspond to the data in each row of your data file.

To create a table, open a command prompt and use sqlcmd.exe to run the following command:

```
sqlcmd.exe -S <server name> -d <database name> -U <username> -P <password> -I -Q "
CREATE TABLE DimDate2
(
 DateId INT NOT NULL,
 CalendarQuarter TINYINT NOT NULL,
 FiscalQuarter TINYINT NOT NULL
)
";
"
```

## 2. Create a source data file

Open Notepad and copy the following lines of data into a new text file and then save this file to your local temp directory, C:\Temp\DimDate2.txt. This data is in ASCII format.

```
20150301,1,3
20150501,2,4
20151001,4,2
20150201,1,3
20151201,4,2
20150801,3,1
20150601,2,4
20151101,4,2
20150401,2,4
20150701,3,1
20150901,3,1
20150101,1,3
```

(Optional) To export your own data from a SQL Server database, open a command prompt and run the following command. Replace TableName, ServerName, DatabaseName, Username, and Password with your own information.

```
bcp <TableName> out C:\Temp\DimDate2_export.txt -S <ServerName> -d <DatabaseName> -U <Username> -P <Password>
-q -c -t ,
```

### 3. Load the data

To load the data, open a command prompt and run the following command, replacing the values for Server Name, Database name, Username, and Password with your own information.

```
bcp DimDate2 in C:\Temp\DimDate2.txt -S <ServerName> -d <DatabaseName> -U <Username> -P <password> -q -c -t ,
```

Use this command to verify the data was loaded properly

```
sqlcmd.exe -S <server name> -d <database name> -U <username> -P <password> -I -Q "SELECT * FROM DimDate2 ORDER
BY 1;"
```

The results should look like this:

DATEID	CALENDARQUARTER	FISCALQUARTER
20150101	1	3
20150201	1	3
20150301	1	3
20150401	2	4
20150501	2	4
20150601	2	4
20150701	3	1
20150801	3	1
20150801	3	1

DATEID	CALENDARQUARTER	FISCALQUARTER
20151001	4	2
20151101	4	2
20151201	4	2

## Next steps

To migrate a SQL Server database, see [SQL Server database migration](#).

# Copy and transform data in Azure SQL Database by using Azure Data Factory

2/4/2020 • 19 minutes to read • [Edit Online](#)

This article outlines how to use Copy Activity in Azure Data Factory to copy data from and to Azure SQL Database, and use Data Flow to transform data in Azure SQL Database. To learn about Azure Data Factory, read the [introductory article](#).

## Supported capabilities

This Azure SQL Database connector is supported for the following activities:

- [Copy activity](#) with [supported source/sink matrix](#) table
- [Mapping data flow](#)
- [Lookup activity](#)
- [GetMetadata activity](#)

For Copy activity, this Azure SQL Database connector supports these functions:

- Copying data by using SQL authentication and Azure Active Directory (Azure AD) Application token authentication with a service principal or managed identities for Azure resources.
- As a source, retrieving data by using a SQL query or a stored procedure.
- As a sink, appending data to a destination table or invoking a stored procedure with custom logic during the copy.

### NOTE

Azure SQL Database [Always Encrypted](#) isn't supported by this connector now. To work around, you can use a [generic ODBC connector](#) and a SQL Server ODBC driver via a self-hosted integration runtime. Follow [this guidance](#) with ODBC driver download and connection string configurations.

### IMPORTANT

If you copy data by using the Azure Data Factory integration runtime, configure an [Azure SQL Server firewall](#) so that Azure services can access the server. If you copy data by using a self-hosted integration runtime, configure the Azure SQL Server firewall to allow the appropriate IP range. This range includes the machine's IP that's used to connect to Azure SQL Database.

## Get started

You can use one of the following tools or SDKs to use the copy activity with a pipeline. Select a link for step-by-step instructions:

- [Copy data tool](#)
- [Azure portal](#)
- [.NET SDK](#)
- [Python SDK](#)
- [Azure PowerShell](#)
- [REST API](#)

- Azure Resource Manager template

The following sections provide details about properties that are used to define Azure Data Factory entities specific to an Azure SQL Database connector.

## Linked service properties

These properties are supported for an Azure SQL Database linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property must be set to <b>AzureSqlDatabase</b> .	Yes
connectionString	Specify information needed to connect to the Azure SQL Database instance for the <b>connectionString</b> property. You also can put a password or service principal key in Azure Key Vault. If it's SQL authentication, pull the <b>password</b> configuration out of the connection string. For more information, see the JSON example following the table and <a href="#">Store credentials in Azure Key Vault</a> .	Yes
servicePrincipalId	Specify the application's client ID.	Yes, when you use Azure AD authentication with a service principal
servicePrincipalKey	Specify the application's key. Mark this field as <b>SecureString</b> to store it securely in Azure Data Factory or <a href="#">reference a secret stored in Azure Key Vault</a> .	Yes, when you use Azure AD authentication with a service principal
tenant	Specify the tenant information, like the domain name or tenant ID, under which your application resides. Retrieve it by hovering the mouse in the upper-right corner of the Azure portal.	Yes, when you use Azure AD authentication with a service principal
connectVia	This <a href="#">integration runtime</a> is used to connect to the data store. You can use the Azure integration runtime or a self-hosted integration runtime if your data store is located in a private network. If not specified, the default Azure integration runtime is used.	No

For different authentication types, refer to the following sections on prerequisites and JSON samples, respectively:

- [SQL authentication](#)
- [Azure AD application token authentication: Service principal](#)
- [Azure AD application token authentication: Managed identities for Azure resources](#)

## TIP

If you hit an error with the error code "UserErrorFailedToConnectToSqlServer" and a message like "The session limit for the database is XXX and has been reached," add `Pooling=false` to your connection string and try again.

## SQL authentication

### Linked service example that uses SQL authentication

```
{
 "name": "AzureSqlDbLinkedService",
 "properties": {
 "type": "AzureSqlDatabase",
 "typeProperties": {
 "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=<databasename>;User ID=<username>@<servername>;Password=<password>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Password in Azure Key Vault

```
{
 "name": "AzureSqlDbLinkedService",
 "properties": {
 "type": "AzureSqlDatabase",
 "typeProperties": {
 "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=<databasename>;User ID=<username>@<servername>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30",
 "password": {
 "type": "AzureKeyVaultSecret",
 "store": {
 "referenceName": "<Azure Key Vault linked service name>",
 "type": "LinkedServiceReference"
 },
 "secretName": "<secretName>"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
 }
}
```

## Service principal authentication

To use a service principal-based Azure AD application token authentication, follow these steps:

1. [Create an Azure Active Directory application](#) from the Azure portal. Make note of the application name and the following values that define the linked service:
  - Application ID
  - Application key
  - Tenant ID
2. [Provision an Azure Active Directory administrator](#) for your Azure SQL Server on the Azure portal if you

haven't already done so. The Azure AD administrator must be an Azure AD user or Azure AD group, but it can't be a service principal. This step is done so that, in the next step, you can use an Azure AD identity to create a contained database user for the service principal.

3. [Create contained database users](#) for the service principal. Connect to the database from or to which you want to copy data by using tools like SQL Server Management Studio, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

```
CREATE USER [your application name] FROM EXTERNAL PROVIDER;
```

4. Grant the service principal needed permissions as you normally do for SQL users or others. Run the following code. For more options, see [this document](#).

```
EXEC sp_addrolemember [role name], [your application name];
```

5. Configure an Azure SQL Database linked service in Azure Data Factory.

#### Linked service example that uses service principal authentication

```
{
 "name": "AzureSqlDbLinkedService",
 "properties": {
 "type": "AzureSqlDatabase",
 "typeProperties": {
 "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=<databasename>;Connection Timeout=30",
 "servicePrincipalId": "<service principal id>",
 "servicePrincipalKey": {
 "type": "SecureString",
 "value": "<service principal key>"
 },
 "tenant": "<tenant info, e.g. microsoft.onmicrosoft.com>"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

#### Managed identities for Azure resources authentication

A data factory can be associated with a [managed identity for Azure resources](#) that represents the specific data factory. You can use this managed identity for Azure SQL Database authentication. The designated factory can access and copy data from or to your database by using this identity.

To use managed identity authentication, follow these steps.

1. [Provision an Azure Active Directory administrator](#) for your Azure SQL Server on the Azure portal if you haven't already done so. The Azure AD administrator can be an Azure AD user or an Azure AD group. If you grant the group with managed identity an admin role, skip steps 3 and 4. The administrator has full access to the database.
2. [Create contained database users](#) for the Azure Data Factory managed identity. Connect to the database from or to which you want to copy data by using tools like SQL Server Management Studio, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

```
CREATE USER [your Data Factory name] FROM EXTERNAL PROVIDER;
```

3. Grant the Data Factory managed identity needed permissions as you normally do for SQL users and others.  
Run the following code. For more options, see [this document](#).

```
EXEC sp_addrolemember [role name], [your Data Factory name];
```

4. Configure an Azure SQL Database linked service in Azure Data Factory.

### Example

```
{
 "name": "AzureSqlDbLinkedService",
 "properties": {
 "type": "AzureSqlDatabase",
 "typeProperties": {
 "connectionString": "Server=tcp:<servername>.database.windows.net,1433;Database=<databasename>;Connection Timeout=30"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Dataset properties

For a full list of sections and properties available to define datasets, see [Datasets](#).

The following properties are supported for Azure SQL Database dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property of the dataset must be set to <b>AzureSqlTable</b> .	Yes
schema	Name of the schema.	No for source, Yes for sink
table	Name of the table/view.	No for source, Yes for sink
tableName	Name of the table/view with schema. This property is supported for backward compatibility. For new workload, use <code>schema</code> and <code>table</code> .	No for source, Yes for sink

### Dataset properties example

```
{
 "name": "AzureSQLDbDataset",
 "properties": {
 {
 "type": "AzureSqlTable",
 "linkedServiceName": {
 "referenceName": "<Azure SQL Database linked service name>",
 "type": "LinkedServiceReference"
 },
 "schema": [< physical schema, optional, retrievable during authoring >],
 "typeProperties": {
 "schema": "<schema_name>",
 "table": "<table_name>"
 }
 }
 }
}
```

## Copy activity properties

For a full list of sections and properties available for defining activities, see [Pipelines](#). This section provides a list of properties supported by the Azure SQL Database source and sink.

### Azure SQL Database as the source

To copy data from Azure SQL Database, the following properties are supported in the copy activity **source** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property of the copy activity source must be set to <b>AzureSqlSource</b> . "SqlSource" type is still supported for backward compatibility.	Yes
sqlReaderQuery	This property uses the custom SQL query to read data. An example is <code>select * from MyTable</code> .	No
sqlReaderStoredProcedureName	The name of the stored procedure that reads data from the source table. The last SQL statement must be a SELECT statement in the stored procedure.	No
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name or value pairs. The names and casing of parameters must match the names and casing of the stored procedure parameters.	No

### Points to note:

- If **sqlReaderQuery** is specified for **AzureSqlSource**, the copy activity runs this query against the Azure SQL Database source to get the data. You also can specify a stored procedure by specifying **sqlReaderStoredProcedureName** and **storedProcedureParameters** if the stored procedure takes parameters.
- If you don't specify either **sqlReaderQuery** or **sqlReaderStoredProcedureName**, the columns defined in the "structure" section of the dataset JSON are used to construct a query. The query `select column1, column2 from mytable` runs against Azure SQL Database. If the dataset definition doesn't have "structure," all columns are selected from the table.

## SQL query example

```
"activities": [
 {
 "name": "CopyFromAzureSQLDatabase",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<Azure SQL Database input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "AzureSqlSource",
 "sqlReaderQuery": "SELECT * FROM MyTable"
 },
 "sink": {
 "type": "<sink type>"
 }
 }
 }
]
```

## Stored procedure example

```
"activities": [
 {
 "name": "CopyFromAzureSQLDatabase",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<Azure SQL Database input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "AzureSqlSource",
 "sqlReaderStoredProcedureName": "CopyTestSrcStoredProcedureWithParameters",
 "storedProcedureParameters": {
 "stringData": { "value": "str3" },
 "identifier": { "value": "$$Text.Format('{0:yyyy}', <datetime parameter>)", "type": "Int" }
 }
 },
 "sink": {
 "type": "<sink type>"
 }
 }
 }
]
```

## Stored procedure definition

```

CREATE PROCEDURE CopyTestSrcStoredProcedureWithParameters
(
 @stringData varchar(20),
 @identifier int
)
AS
SET NOCOUNT ON;
BEGIN
 select *
 from dbo.UnitTestingSrcTable
 where dbo.UnitTestingSrcTable.stringData != stringData
 and dbo.UnitTestingSrcTable.identifier != identifier
END
GO

```

## Azure SQL Database as the sink

### TIP

Learn more about the supported write behaviors, configurations, and best practices from [Best practice for loading data into Azure SQL Database](#).

To copy data to Azure SQL Database, the following properties are supported in the copy activity **sink** section:

PROPERTY	DESCRIPTION	REQUIRED
type	The <b>type</b> property of the copy activity sink must be set to <b>AzureSqlSink</b> . "SqlSink" type is still supported for backward compatibility.	Yes
writeBatchSize	Number of rows to insert into the SQL table <i>per batch</i> . The allowed value is <b>integer</b> (number of rows). By default, Azure Data Factory dynamically determines the appropriate batch size based on the row size.	No
writeBatchTimeout	The wait time for the batch insert operation to finish before it times out. The allowed value is <b>timespan</b> . An example is "00:30:00" (30 minutes).	No
preCopyScript	Specify a SQL query for the copy activity to run before writing data into Azure SQL Database. It's invoked only once per copy run. Use this property to clean up the preloaded data.	No
sqlWriterStoredProcedureName	The name of the stored procedure that defines how to apply source data into a target table. This stored procedure is <i>invoked per batch</i> . For operations that run only once and have nothing to do with source data, for example, delete or truncate, use the <b>preCopyScript</b> property.	No

PROPERTY	DESCRIPTION	REQUIRED
storedProcedureTableTypeParameterName	The parameter name of the table type specified in the stored procedure.	No
sqlWriterTableType	The table type name to be used in the stored procedure. The copy activity makes the data being moved available in a temp table with this table type. Stored procedure code can then merge the data that's being copied with existing data.	No
storedProcedureParameters	Parameters for the stored procedure. Allowed values are name and value pairs. Names and casing of parameters must match the names and casing of the stored procedure parameters.	No
tableOption	Specifies whether to automatically create the sink table if not exists based on the source schema. Auto table creation is not supported when sink specifies stored procedure or staged copy is configured in copy activity. Allowed values are: <code>none</code> (default), <code>autoCreate</code> .	No
disableMetricsCollection	Data Factory collects metrics such as Azure SQL Database DTUs for copy performance optimization and recommendations. If you are concerned with this behavior, specify <code>true</code> to turn it off.	No (default is <code>false</code> )

### Example 1: Append data

```
"activities": [
 {
 "name": "CopyToAzureSQLDatabase",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<Azure SQL Database output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "<source type>"
 },
 "sink": {
 "type": "AzureSqlSink",
 "writeBatchSize": 100000,
 "tableOption": "autoCreate"
 }
 }
 }
]
```

### Example 2: Invoke a stored procedure during copy

Learn more details from [Invoke a stored procedure from a SQL sink](#).

```

"activities": [
 {
 "name": "CopyToAzureSQLDatabase",
 "type": "Copy",
 "inputs": [
 {
 "referenceName": "<input dataset name>",
 "type": "DatasetReference"
 }
],
 "outputs": [
 {
 "referenceName": "<Azure SQL Database output dataset name>",
 "type": "DatasetReference"
 }
],
 "typeProperties": {
 "source": {
 "type": "<source type>"
 },
 "sink": {
 "type": "AzureSqlSink",
 "sqlWriterStoredProcedureName": "CopyTestStoredProcedureWithParameters",
 "storedProcedureTableTypeParameterName": "MyTable",
 "sqlWriterTableType": "MyTableType",
 "storedProcedureParameters": {
 "identifier": { "value": "1", "type": "Int" },
 "stringData": { "value": "str1" }
 }
 }
 }
 }
]

```

## Best practice for loading data into Azure SQL Database

When you copy data into Azure SQL Database, you might require different write behavior:

- **Append:** My source data has only new records.
- **Upsert:** My source data has both inserts and updates.
- **Overwrite:** I want to reload an entire dimension table each time.
- **Write with custom logic:** I need extra processing before the final insertion into the destination table.

Refer to the respective sections about how to configure in Azure Data Factory and best practices.

### Append data

Appending data is the default behavior of this Azure SQL Database sink connector. Azure Data Factory does a bulk insert to write to your table efficiently. You can configure the source and sink accordingly in the copy activity.

### Upsert data

**Option 1:** When you have a large amount of data to copy, use the following approach to do an upsert:

- First, use a **database scoped temporary table** to bulk load all records by using the copy activity. Because operations against database scoped temporary tables aren't logged, you can load millions of records in seconds.
- Run a stored procedure activity in Azure Data Factory to apply a **MERGE** or **INSERT/UPDATE** statement. Use the temp table as the source to perform all updates or inserts as a single transaction. In this way, the number of round trips and log operations is reduced. At the end of the stored procedure activity, the temp table can be truncated to be ready for the next upsert cycle.

As an example, in Azure Data Factory, you can create a pipeline with a **Copy activity** chained with a **Stored Procedure activity**. The former copies data from your source store into an Azure SQL Database temporary table, for example, **##UpsertTempTable**, as the table name in the dataset. Then the latter invokes a stored procedure to merge source data from the temp table into the target table and clean up the temp table.



In your database, define a stored procedure with MERGE logic, like the following example, which is pointed to from the previous stored procedure activity. Assume that the target is the **Marketing** table with three columns: **ProfileID**, **State**, and **Category**. Do the upsert based on the **ProfileID** column.

```
CREATE PROCEDURE [dbo].[spMergeData]
AS
BEGIN
 MERGE TargetTable AS target
 USING ##UpsertTempTable AS source
 ON (target.[ProfileID] = source.[ProfileID])
 WHEN MATCHED THEN
 UPDATE SET State = source.State
 WHEN NOT matched THEN
 INSERT ([ProfileID], [State], [Category])
 VALUES (source.ProfileID, source.State, source.Category);

 TRUNCATE TABLE ##UpsertTempTable
END
```

**Option 2:** You also can choose to [invoke a stored procedure within the copy activity](#). This approach runs each row in the source table instead of using bulk insert as the default approach in the copy activity, which isn't appropriate for large-scale upsert.

### Overwrite the entire table

You can configure the **preCopyScript** property in the copy activity sink. In this case, for each copy activity that runs, Azure Data Factory runs the script first. Then it runs the copy to insert the data. For example, to overwrite the entire table with the latest data, specify a script to first delete all the records before you bulk load the new data from the source.

### Write data with custom logic

The steps to write data with custom logic are similar to those described in the [Upsert data](#) section. When you need to apply extra processing before the final insertion of source data into the destination table, for large scale, you can do one of two things:

- Load to a database scoped temporary table and then invoke a stored procedure.
- Invoke a stored procedure during copy.

## Invoke a stored procedure from a SQL sink

When you copy data into Azure SQL Database, you also can configure and invoke a user-specified stored procedure with additional parameters. The stored procedure feature takes advantage of [table-valued parameters](#).

**TIP**

Invoking a stored procedure processes the data row by row instead of by using a bulk operation, which we don't recommend for large-scale copy. Learn more from [Best practice for loading data into Azure SQL Database](#).

You can use a stored procedure when built-in copy mechanisms don't serve the purpose. An example is when you want to apply extra processing before the final insertion of source data into the destination table. Some extra processing examples are when you want to merge columns, look up additional values, and insert into more than one table.

The following sample shows how to use a stored procedure to do an upsert into a table in Azure SQL Database. Assume that the input data and the sink **Marketing** table each have three columns: **ProfileID**, **State**, and **Category**. Do the upsert based on the **ProfileID** column, and only apply it for a specific category called "ProductA".

1. In your database, define the table type with the same name as **sqlWriterTableType**. The schema of the table type is the same as the schema returned by your input data.

```
CREATE TYPE [dbo].[MarketingType] AS TABLE(
 [ProfileID] [varchar](256) NOT NULL,
 [State] [varchar](256) NOT NULL,
 [Category] [varchar](256) NOT NULL
)
```

2. In your database, define the stored procedure with the same name as **sqlWriterStoredProcName**. It handles input data from your specified source and merges into the output table. The parameter name of the table type in the stored procedure is the same as **tableName** defined in the dataset.

```
CREATE PROCEDURE spOverwriteMarketing @Marketing [dbo].[MarketingType] READONLY, @category varchar(256)
AS
BEGIN
MERGE [dbo].[Marketing] AS target
USING @Marketing AS source
ON (target.ProfileID = source.ProfileID and target.Category = @category)
WHEN MATCHED THEN
 UPDATE SET State = source.State
WHEN NOT MATCHED THEN
 INSERT (ProfileID, State, Category)
 VALUES (source.ProfileID, source.State, source.Category);
END
```

3. In Azure Data Factory, define the **SQL sink** section in the copy activity as follows:

```
"sink": {
 "type": "AzureSqlSink",
 "sqlWriterStoredProcName": "spOverwriteMarketing",
 "storedProcTableTypeParameterName": "Marketing",
 "sqlWriterTableType": "MarketingType",
 "storedProcParameters": {
 "category": {
 "value": "ProductA"
 }
 }
}
```

## Mapping data flow properties

When transforming data in mapping data flow, you can read and write to tables from Azure SQL Database. For more information, see the [source transformation](#) and [sink transformation](#) in mapping data flows.

## Source transformation

Settings specific to Azure SQL Database are available in the **Source Options** tab of the source transformation.

**Input:** Select whether you point your source at a table (equivalent of `Select * from <table-name>`) or enter a custom SQL query.

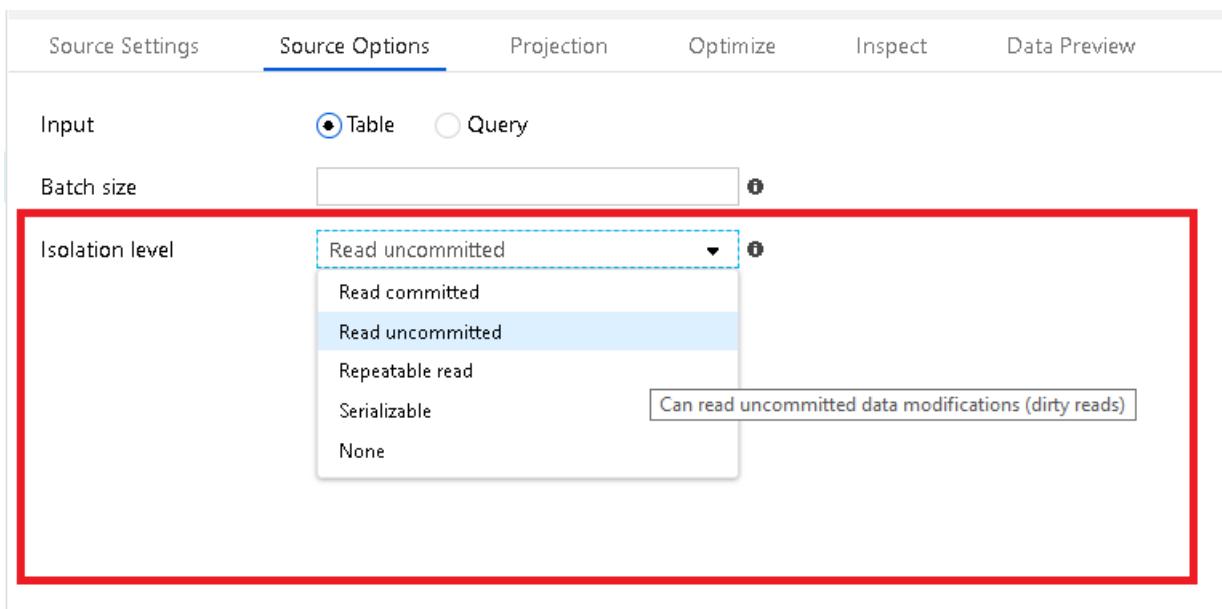
**Query:** If you select Query in the input field, enter a SQL query for your source. This setting overrides any table that you've chosen in the dataset. **Order By** clauses aren't supported here, but you can set a full SELECT FROM statement. You can also use user-defined table functions. `select * from udfGetData()` is a UDF in SQL that returns a table. This query will produce a source table that you can use in your data flow. Using queries is also a great way to reduce rows for testing or for lookups.

- SQL Example: `Select * from MyTable where customerId > 1000 and customerId < 2000`

**Batch size:** Enter a batch size to chunk large data into reads.

**Isolation Level:** The default for SQL sources in mapping data flow is read uncommitted. You can change the isolation level here to one of these values:

- Read Committed
- Read Uncommitted
- Repeatable Read
- Serializable
- None (ignore isolation level)



## Sink transformation

Settings specific to Azure SQL Database are available in the **Settings** tab of the sink transformation.

**Update method:** Determines what operations are allowed on your database destination. The default is to only allow inserts. To update, upsert, or delete rows, an alter-row transformation is required to tag rows for those actions. For updates, upserts and deletes, a key column or columns must be set to determine which row to alter.

Sink Settings Mapping Optimize Inspect Data preview

Allowing delete, upsert, or update requires an Alter Row transformation to set row policies. Click to insert an Alter Row transformation. [Add Alter Row](#)

Update method

Allow insert  
 Allow delete  
 Allow upsert  
 Allow update

Key columns \* [?](#)

Skip writing key columns

The column name that you pick as the key here will be used by ADF as part of the subsequent update, upsert, delete. Therefore, you must pick a column that exists in the Sink mapping. If you wish to not write the value to this key column, then click "Skip writing key columns".

**Table action:** Determines whether to recreate or remove all rows from the destination table prior to writing.

- None: No action will be done to the table.
- Recreate: The table will get dropped and recreated. Required if creating a new table dynamically.
- Truncate: All rows from the target table will get removed.

**Batch size:** Controls how many rows are being written in each bucket. Larger batch sizes improve compression and memory optimization, but risk out of memory exceptions when caching data.

**Pre and Post SQL scripts:** Enter multi-line SQL scripts that will execute before (pre-processing) and after (post-processing) data is written to your Sink database

Sink Settings Mapping Optimize Inspect Data preview

Update method

Allow insert  
 Allow delete  
 Allow upsert  
 Allow update

Table action

None  Recreate table  Truncate table

Batch size

Pre SQL scripts

```
set IDENTITY_INSERT mytable on
```

Post SQL scripts

```
set IDENTITY_INSERT mytable off
```

Add dynamic content [Alt+P]

## Data type mapping for Azure SQL Database

When data is copied from or to Azure SQL Database, the following mappings are used from Azure SQL Database data types to Azure Data Factory interim data types. To learn how the copy activity maps the source schema and data type to the sink, see [Schema and data type mappings](#).

AZURE SQL DATABASE DATA TYPE	AZURE DATA FACTORY INTERIM DATA TYPE
bigint	Int64
binary	Byte[]
bit	Boolean
char	String, Char[]
date	DateTime
Datetime	DateTime
datetime2	DateTime
Datetimeoffset	DateTimeOffset
Decimal	Decimal
FILESTREAM attribute (varbinary(max))	Byte[]
Float	Double
image	Byte[]
int	Int32
money	Decimal
nchar	String, Char[]
ntext	String, Char[]
numeric	Decimal
nvarchar	String, Char[]
real	Single
rowversion	Byte[]
smalldatetime	DateTime
smallint	Int16
smallmoney	Decimal

AZURE SQL DATABASE DATA TYPE	AZURE DATA FACTORY INTERIM DATA TYPE
sql_variant	Object
text	String, Char[]
time	TimeSpan
timestamp	Byte[]
tinyint	Byte
uniqueidentifier	Guid
varbinary	Byte[]
varchar	String, Char[]
xml	Xml

**NOTE**

For data types that map to the Decimal interim type, currently Azure Data Factory supports precision up to 28. If you have data with precision larger than 28, consider converting to a string in SQL query.

## Lookup activity properties

To learn details about the properties, check [Lookup activity](#).

## GetMetadata activity properties

To learn details about the properties, check [GetMetadata activity](#)

## Next steps

For a list of data stores supported as sources and sinks by the copy activity in Azure Data Factory, see [Supported data stores and formats](#).

# SQL Database application development overview

11/14/2019 • 2 minutes to read • [Edit Online](#)

This article walks through the basic considerations that a developer should be aware of when writing code to connect to Azure SQL Database. This article applies to all deployment models of Azure SQL Database (Single database, Elastic pools, Managed instance).

## TIP

Look at the getting started guides for [single databases](#) and [managed instances](#) if you need to setup your Azure SQL Database.

## Language and platform

You can use various [programming languages and platforms](#) to connect and query Azure SQL Database. You can find [sample applications](#) that you can use to connect to the Azure SQL Database.

You can leverage open-source tools like [cheetah](#), [sql-cli](#), [VS Code](#). Additionally, Azure SQL Database works with Microsoft tools like [Visual Studio](#) and [SQL Server Management Studio](#). You can also use the Azure portal, PowerShell, and REST APIs help you gain additional productivity.

## Authentication

Access to Azure SQL Database is protected with logins and firewalls. Azure SQL Database supports both SQL Server and [Azure Active Directory \(AAD\) authentication](#) users and logins. AAD logins are available only in Managed Instance.

Learn more about [managing database access and login](#).

## Connections

In your client connection logic, override the default timeout to be 30 seconds. The default of 15 seconds is too short for connections that depend on the internet.

If you are using a [connection pool](#), be sure to close the connection the instant your program is not actively using it, and is not preparing to reuse it.

Avoid long-running transactions because any infrastructure or connection failure might roll back the transaction. If possible, split the transaction in the multiple smaller transactions and use [batching to improve performance](#).

## Resiliency

Azure SQL Database is a cloud service where you might expect transient errors that happen in the underlying infrastructure or in the communication between cloud entities. Although Azure SQL Database is resilient on the transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while connecting to SQL Database, your code should [retry the call](#). We recommend that retry logic use backoff logic, so that it does not overwhelm the SQL Database with multiple clients retrying simultaneously. Retry logic depends on the [error messages for SQL Database client programs](#).

For more information about how to prepare for planned maintenance events on your Azure SQL database, see [planning for Azure maintenance events in Azure SQL Database](#).

## Network considerations

- On the computer that hosts your client program, ensure the firewall allows outgoing TCP communication on port 1433. More information: [Configure an Azure SQL Database firewall](#).
- If your client program connects to SQL Database while your client runs on an Azure virtual machine (VM), you must open certain port ranges on the VM. More information: [Ports beyond 1433 for ADO.NET 4.5 and SQL Database](#).
- Client connections to Azure SQL Database sometimes bypass the proxy and interact directly with the database. Ports other than 1433 become important. For more information, [Azure SQL Database connectivity architecture](#) and [Ports beyond 1433 for ADO.NET 4.5 and SQL Database](#).
- For networking configuration for a managed instance, see [network configuration for managed instances](#).

## Next steps

Explore all the [capabilities of SQL Database](#).

# Getting started with JSON features in Azure SQL Database

11/7/2019 • 6 minutes to read • [Edit Online](#)

Azure SQL Database lets you parse and query data represented in JavaScript Object Notation ([JSON](#)) format, and export your relational data as JSON text. The following JSON scenarios are available in Azure SQL Database:

- [Formatting relational data in JSON format](#) using `FOR JSON` clause.
- [Working with JSON data](#)
- [Querying JSON data](#) using JSON scalar functions.
- [Transforming JSON into tabular format](#) using `OPENJSON` function.

## Formatting relational data in JSON format

If you have a web service that takes data from the database layer and provides a response in JSON format, or client-side JavaScript frameworks or libraries that accept data formatted as JSON, you can format your database content as JSON directly in a SQL query. You no longer have to write application code that formats results from Azure SQL Database as JSON, or include some JSON serialization library to convert tabular query results and then serialize objects to JSON format. Instead, you can use the `FOR JSON` clause to format SQL query results as JSON in Azure SQL Database and use it directly in your application.

In the following example, rows from the `Sales.Customer` table are formatted as JSON by using the `FOR JSON` clause:

```
select CustomerName, PhoneNumber, FaxNumber
from Sales.Customers
FOR JSON PATH
```

The `FOR JSON PATH` clause formats the results of the query as JSON text. Column names are used as keys, while the cell values are generated as JSON values:

```
[{"CustomerName": "Eric Torres", "PhoneNumber": "(307) 555-0100", "FaxNumber": "(307) 555-0101"}, {"CustomerName": "Cosmina Vlad", "PhoneNumber": "(505) 555-0100", "FaxNumber": "(505) 555-0101"}, {"CustomerName": "Bala Dixit", "PhoneNumber": "(209) 555-0100", "FaxNumber": "(209) 555-0101"}]
```

The result set is formatted as a JSON array where each row is formatted as a separate JSON object.

`PATH` indicates that you can customize the output format of your JSON result by using dot notation in column aliases. The following query changes the name of the `"CustomerName"` key in the output JSON format, and puts phone and fax numbers in the `"Contact"` sub-object:

```
select CustomerName as Name, PhoneNumber as [Contact.Phone], FaxNumber as [Contact.Fax]
from Sales.Customers
where CustomerID = 931
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

The output of this query looks like this:

```
{
 "Name": "Nada Jovanovic",
 "Contact": {
 "Phone": "(215) 555-0100",
 "Fax": "(215) 555-0101"
 }
}
```

In this example, we returned a single JSON object instead of an array by specifying the [WITHOUT\\_ARRAY\\_WRAPPER](#) option. You can use this option if you know that you are returning a single object as a result of query.

The main value of the FOR JSON clause is that it lets you return complex hierarchical data from your database formatted as nested JSON objects or arrays. The following example shows how to include the rows from the `Orders` table that belong to the `Customer` as a nested array of `Orders`:

```
select CustomerName as Name, PhoneNumber as Phone, FaxNumber as Fax,
 Orders.OrderID, Orders.OrderDate, Orders.ExpectedDeliveryDate
 from Sales.Customers Customer
 join Sales.Orders Orders
 on Customer.CustomerID = Orders.CustomerID
 where Customer.CustomerID = 931
FOR JSON AUTO, WITHOUT_ARRAY_WRAPPER
```

Instead of sending separate queries to get Customer data and then to fetch a list of related Orders, you can get all the necessary data with a single query, as shown in the following sample output:

```
{
 "Name": "Nada Jovanovic",
 "Phone": "(215) 555-0100",
 "Fax": "(215) 555-0101",
 "Orders": [
 {"OrderID": 382, "OrderDate": "2013-01-07", "ExpectedDeliveryDate": "2013-01-08"},
 {"OrderID": 395, "OrderDate": "2013-01-07", "ExpectedDeliveryDate": "2013-01-08"},
 {"OrderID": 1657, "OrderDate": "2013-01-31", "ExpectedDeliveryDate": "2013-02-01"}
]
}
```

## Working with JSON data

If you don't have strictly structured data, if you have complex sub-objects, arrays, or hierarchical data, or if your data structures evolve over time, the JSON format can help you to represent any complex data structure.

JSON is a textual format that can be used like any other string type in Azure SQL Database. You can send or store JSON data as a standard NVARCHAR:

```

CREATE TABLE Products (
 Id int identity primary key,
 Title nvarchar(200),
 Data nvarchar(max)
)
go
CREATE PROCEDURE InsertProduct(@title nvarchar(200), @json nvarchar(max))
AS BEGIN
 insert into Products(Title, Data)
 values(@title, @json)
END

```

The JSON data used in this example is represented by using the NVARCHAR(MAX) type. JSON can be inserted into this table or provided as an argument of the stored procedure using standard Transact-SQL syntax as shown in the following example:

```
EXEC InsertProduct 'Toy car', '{"Price":50,"Color":"White","tags":["toy","children","games"]}'
```

Any client-side language or library that works with string data in Azure SQL Database will also work with JSON data. JSON can be stored in any table that supports the NVARCHAR type, such as a Memory-optimized table or a System-versioned table. JSON does not introduce any constraint either in the client-side code or in the database layer.

## Querying JSON data

If you have data formatted as JSON stored in Azure SQL tables, JSON functions let you use this data in any SQL query.

JSON functions that are available in Azure SQL database let you treat data formatted as JSON as any other SQL data type. You can easily extract values from the JSON text, and use JSON data in any query:

```

select Id, Title, JSON_VALUE(Data, '$.Color'), JSON_QUERY(Data, '$.tags')
from Products
where JSON_VALUE(Data, '$.Color') = 'White'

update Products
set Data = JSON_MODIFY(Data, '$.Price', 60)
where Id = 1

```

The JSON\_VALUE function extracts a value from JSON text stored in the Data column. This function uses a JavaScript-like path to reference a value in JSON text to extract. The extracted value can be used in any part of SQL query.

The JSON\_QUERY function is similar to JSON\_VALUE. Unlike JSON\_VALUE, this function extracts complex sub-object such as arrays or objects that are placed in JSON text.

The JSON MODIFY function lets you specify the path of the value in the JSON text that should be updated, as well as a new value that will overwrite the old one. This way you can easily update JSON text without reparsing the entire structure.

Since JSON is stored in a standard text, there are no guarantees that the values stored in text columns are properly formatted. You can verify that text stored in JSON column is properly formatted by using standard Azure SQL Database check constraints and the ISJSON function:

```

ALTER TABLE Products
ADD CONSTRAINT [Data should be formatted as JSON]
CHECK (ISJSON(Data) > 0)

```

If the input text is properly formatted JSON, the ISJSON function returns the value 1. On every insert or update of JSON column, this constraint will verify that new text value is not malformed JSON.

## Transforming JSON into tabular format

Azure SQL Database also lets you transform JSON collections into tabular format and load or query JSON data.

OPENJSON is a table-value function that parses JSON text, locates an array of JSON objects, iterates through the elements of the array, and returns one row in the output result for each element of the array.

JSON input:

```
{
 "Orders": [
 {
 "Order": {
 "Number": "SO43659",
 "Date": "2011-05-31T00:00:00"
 },
 "Account": "Microsoft",
 "Item": {
 "Price": 59.99,
 "Quantity": 1
 }
 },
 {
 "Order": {
 "Number": "SO43661",
 "Date": "2011-06-01T00:00:00"
 },
 "Account": "Nokia",
 "Item": {
 "Price": 24.99,
 "Quantity": 3
 }
 }
]
}
```

Query with OPENJSON function:

```

SELECT *
FROM OPENJSON (@json, N'$.[].Orders')
WITH (
 Number varchar(200) N'$.Order.Number',
 Date datetime N'$.Order.Date',
 Customer varchar(200) N'$.Account',
 Quantity int N'$.Item.Quantity'
)

```

Output table data:

Number	Date	Customer	Quantity
SO43659	2011-05-31T00:00:00	Microsoft	1
SO43661	2011-06-01T00:00:00	Nokia	3

In the example above, we can specify where to locate the JSON array that should be opened (in the \$.Orders path), what columns should be returned as result, and where to find the JSON values that will be returned as cells.

We can transform a JSON array in the @orders variable into a set of rows, analyze this result set, or insert rows into a standard table:

```

CREATE PROCEDURE InsertOrders(@orders nvarchar(max))
AS BEGIN

 insert into Orders(Number, Date, Customer, Quantity)
 select Number, Date, Customer, Quantity
 FROM OPENJSON (@orders)
 WITH (
 Number varchar(200),
 Date datetime,
 Customer varchar(200),
 Quantity int
)
END

```

The collection of orders formatted as a JSON array and provided as a parameter to the stored procedure can be parsed and inserted into the Orders table.

## Next steps

To learn how to integrate JSON into your application, check out these resources:

- [TechNet Blog](#)
- [MSDN documentation](#)
- [Channel 9 video](#)

To learn about various scenarios for integrating JSON into your application, see the demos in this [Channel 9 video](#) or find a scenario that matches your use case in [JSON Blog posts](#).

# Use Azure Functions to connect to an Azure SQL Database

12/10/2019 • 4 minutes to read • [Edit Online](#)

This article shows you how to use Azure Functions to create a scheduled job that connects to an Azure SQL Database or Azure SQL Managed Instance. The function code cleans up rows in a table in the database. The new C# function is created based on a pre-defined timer trigger template in Visual Studio 2019. To support this scenario, you must also set a database connection string as an app setting in the function app. For Azure SQL Managed Instance you need to [enable public endpoint](#) to be able to connect from Azure Functions. This scenario uses a bulk operation against the database.

If this is your first experience working with C# Functions, you should read the [Azure Functions C# developer reference](#).

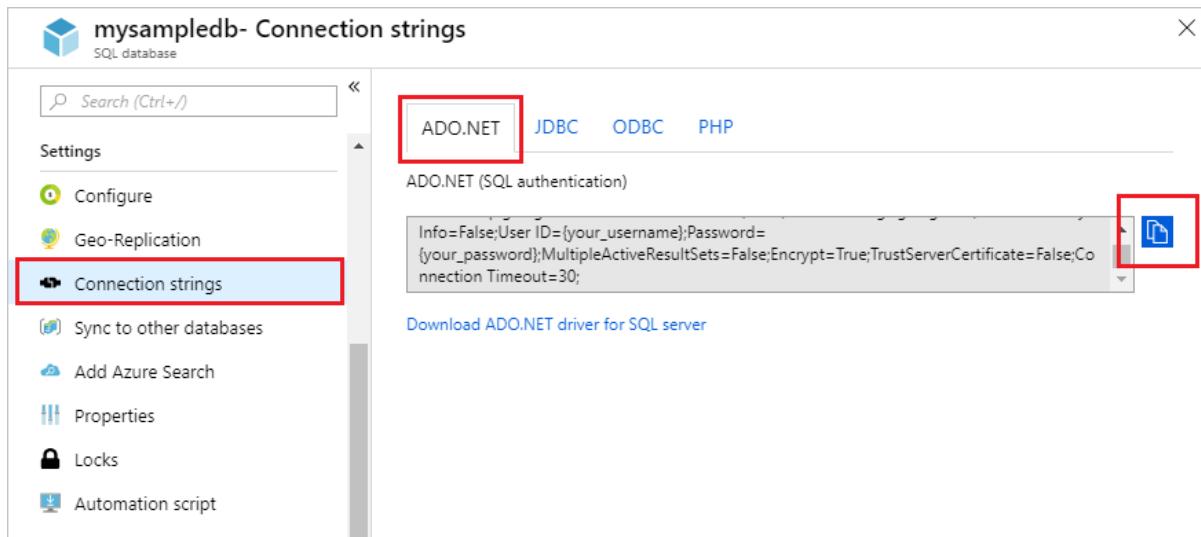
## Prerequisites

- Complete the steps in the article [Create your first function using Visual Studio](#) to create a local function app that targets version 2.x or a later version of the runtime. You must also have published your project to a function app in Azure.
- This article demonstrates a Transact-SQL command that executes a bulk cleanup operation in the **SalesOrderHeader** table in the AdventureWorksLT sample database. To create the AdventureWorksLT sample database, complete the steps in the article [Create an Azure SQL database in the Azure portal](#).
- You must add a [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart. This rule is required to be able access the SQL database instance from your local computer.

## Get connection information

You need to get the connection string for the database you created when you completed [Create an Azure SQL database in the Azure portal](#).

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and select your database on the **SQL databases** page.
3. Select **Connection strings** under **Settings** and copy the complete **ADO.NET** connection string. For Azure SQL Managed Instance copy connection string for public endpoint.



## Set the connection string

A function app hosts the execution of your functions in Azure. As a best security practice, store connection strings and other secrets in your function app settings. Using application settings prevents accidental disclosure of the connection string with your code. You can access app settings for your function app right from Visual Studio.

You must have previously published your app to Azure. If you haven't already done so, [Publish your function app to Azure](#).

1. In Solution Explorer, right-click the function app project and choose **Publish > Manage application settings....** Select **Add setting**, in **New app setting name**, type `sqldb_connection`, and select **OK**.

Publish

Publish your app to Azure or another host. [Learn more](#)

SqlConnectionArticle20181029112739 - Zip Deploy ▾ Publish

New Profile... Actions▼

Site URL https://sqlconnectionarticl... Manage Application Settings...  
Configuration Release Manage Profile Settings...

**Application Settings**

FUNCTIONS\_WORKER\_RUNTIME

Local	do	X
Remote	New App Setting Name Insert value from Local	X
AzureWebJob		X
Local	Us	
Remote	De	

OK Cancel

Insert value from Local

FUNCTIONS\_EXTENSION\_VERSION

+ Add Setting OK Cancel

The screenshot shows the 'Application Settings' dialog for an Azure Function. It lists settings like 'FUNCTIONS\_WORKER\_RUNTIME' and 'FUNCTIONS\_EXTENSION\_VERSION'. A new setting 'sqldb\_connection' is being added under 'FUNCTIONS\_WORKER\_RUNTIME'. The 'Remote' field for this setting is currently empty. The 'Insert value from Local' link below the 'Local' field is highlighted with a red box.

2. In the new **sqldb\_connection** setting, paste the connection string you copied in the previous section into the **Local** field and replace `{your_username}` and `{your_password}` placeholders with real values. Select **Insert value from local** to copy the updated value into the **Remote** field, and then select **OK**.

**Application Settings**

WEBSITE\_RUN\_FROM\_PACKAGE

Local		X
Remote	1	

Insert value from Local

sqldb\_connection

Local*	Server=tcp:mysqlserver.database.windows.net,1433;Initial Catalog=mysql;Persist Se	X
Remote*	Server=tcp:mysqlserver.database.windows.net,1433;Initial Catalog=mysql;Persist Se	

Insert value from Local

+ Add Setting OK Cancel

The screenshot shows the 'Application Settings' dialog again, this time with the 'sqldb\_connection' setting selected. The 'Local' field contains a connection string: 'Server=tcp:mysqlserver.database.windows.net,1433;Initial Catalog=mysql;Persist Se...'. The 'Remote' field also contains the same connection string. Both fields have a red box around them. The 'Insert value from Local' link below the 'Local' field is highlighted with a red box. The 'OK' button at the bottom right is also highlighted with a red box.

The connection strings are stored encrypted in Azure (**Remote**). To prevent leaking secrets, the local.settings.json project file (**Local**) should be excluded from source control, such as by using a .gitignore file.

## Add the SqlClient package to the project

You need to add the NuGet package that contains the `SqlClient` library. This data access library is needed to connect to a SQL database.

1. Open your local function app project in Visual Studio 2019.
2. In Solution Explorer, right-click the function app project and choose **Manage NuGet Packages**.
3. On the **Browse** tab, search for `System.Data.SqlClient` and, when found, select it.
4. In the **System.Data.SqlClient** page, select version `4.5.1` and then click **Install**.
5. When the install completes, review the changes and then click **OK** to close the **Preview** window.
6. If a **License Acceptance** window appears, click **I Accept**.

Now, you can add the C# function code that connects to your SQL Database.

## Add a timer triggered function

1. In Solution Explorer, right-click the function app project and choose **Add > New Azure function**.
2. With the **Azure Functions** template selected, name the new item something like `DatabaseCleanup.cs` and select **Add**.
3. In the **New Azure function** dialog box, choose **Timer trigger** and then **OK**. This dialog creates a code file for the timer triggered function.
4. Open the new code file and add the following using statements at the top of the file:

```
using System.Data.SqlClient;
using System.Threading.Tasks;
```

5. Replace the existing `Run` function with the following code:

```
[FunctionName("DatabaseCleanup")]
public static async Task Run([TimerTrigger("*/15 * * * *")]TimerInfo myTimer, ILogger log)
{
 // Get the connection string from app settings and use it to create a connection.
 var str = Environment.GetEnvironmentVariable("sqlDb_Connection");
 using (SqlConnection conn = new SqlConnection(str))
 {
 conn.Open();
 var text = "UPDATE SalesLT.SalesOrderHeader " +
 "SET [Status] = 5 WHERE ShipDate < GetDate();";

 using (SqlCommand cmd = new SqlCommand(text, conn))
 {
 // Execute the command and log the # rows affected.
 var rows = await cmd.ExecuteNonQueryAsync();
 log.LogInformation($"{rows} rows were updated");
 }
 }
}
```

This function runs every 15 seconds to update the `Status` column based on the ship date. To learn more about the Timer trigger, see [Timer trigger for Azure Functions](#).

6. Press **F5** to start the function app. The [Azure Functions Core Tools](#) execution window opens behind Visual Studio.
7. At 15 seconds after startup, the function runs. Watch the output and note the number of rows updated in the **SalesOrderHeader** table.

The screenshot shows the command-line interface for Azure Functions Core Tools. It displays log messages from the function's execution. A red box highlights the message "[10/29/2018 10:53:09 PM] 32 rows were updated".

```
C:\AzureFunctionsTools\Releases\2.10.1\cli\func.exe
[10/29/2018 10:52:49 PM] SqlConnectionArticle.Function1.Run
[10/29/2018 10:52:49 PM]
[10/29/2018 10:52:49 PM] Host initialized (467ms)
[10/29/2018 10:52:49 PM] The next 5 occurrences of the 'SqlConnectionArticle.DatabaseCleanup.Run' schedule will be:
[10/29/2018 10:52:49 PM] 10/29/2018 3:53:00 PM
[10/29/2018 10:52:49 PM] 10/29/2018 3:53:15 PM
[10/29/2018 10:52:49 PM] 10/29/2018 3:53:30 PM
[10/29/2018 10:52:49 PM] 10/29/2018 3:53:45 PM
[10/29/2018 10:52:49 PM] 10/29/2018 3:54:00 PM
[10/29/2018 10:52:49 PM]
[10/29/2018 10:52:49 PM] Host started (821ms)
[10/29/2018 10:52:49 PM] Job host started
Hosting environment: Production
Content root path: C:\source\repos\SqlConnectionArticle\SqlConnectionArticle\bin\Debug\netcoreapp2.1
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.
Listening on http://0.0.0.0:7071/
Hit CTRL-C to exit...

Http Functions:

 Function1: [GET,POST] http://localhost:7071/api/Function1

[10/29/2018 10:52:54 PM] Host lock lease acquired by instance ID '0000000000000000000000006C79E40E'.
[10/29/2018 10:53:00 PM] Executing 'DatabaseCleanup' (Reason='Timer fired at 2018-10-29T15:53:00.0271201-07:00', Id=2
67f6418-ddfb-4f5c-a065-5575618ca147)
[10/29/2018 10:53:09 PM] 32 rows were updated
```

On the first execution, you should update 32 rows of data. Following runs update no data rows, unless you make changes to the **SalesOrderHeader** table data so that more rows are selected by the `UPDATE` statement.

If you plan to [publish this function](#), remember to change the `TimerTrigger` attribute to a more reasonable [cron schedule](#) than every 15 seconds.

## Next steps

Next, learn how to use Functions with Logic Apps to integrate with other services.

[Create a function that integrates with Logic Apps](#)

For more information about Functions, see the following articles:

- [Azure Functions developer reference](#)  
Programmer reference for coding functions and defining triggers and bindings.
- [Testing Azure Functions](#)  
Describes various tools and techniques for testing your functions.

# Accelerate real-time big data analytics with Spark connector for Azure SQL Database and SQL Server

2/19/2020 • 4 minutes to read • [Edit Online](#)

The Spark connector for Azure SQL Database and SQL Server enables SQL databases, including Azure SQL Database and SQL Server, to act as input data source or output data sink for Spark jobs. It allows you to utilize real-time transactional data in big data analytics and persist results for adhoc queries or reporting. Compared to the built-in JDBC connector, this connector provides the ability to bulk insert data into SQL databases. It can outperform row by row insertion with 10x to 20x faster performance. The Spark connector for Azure SQL Database and SQL Server also supports AAD authentication. It allows you securely connecting to your Azure SQL database from Azure Databricks using your AAD account. It provides similar interfaces with the built-in JDBC connector. It is easy to migrate your existing Spark jobs to use this new connector.

## Download

To get started, download the Spark to SQL DB connector from the [azure-sqldb-spark repository](#) on GitHub.

## Official Supported Versions

COMPONENT	VERSION
Apache Spark	2.0.2 or later
Scala	2.10 or later
Microsoft JDBC Driver for SQL Server	6.2 or later
Microsoft SQL Server	SQL Server 2008 or later
Azure SQL Database	Supported

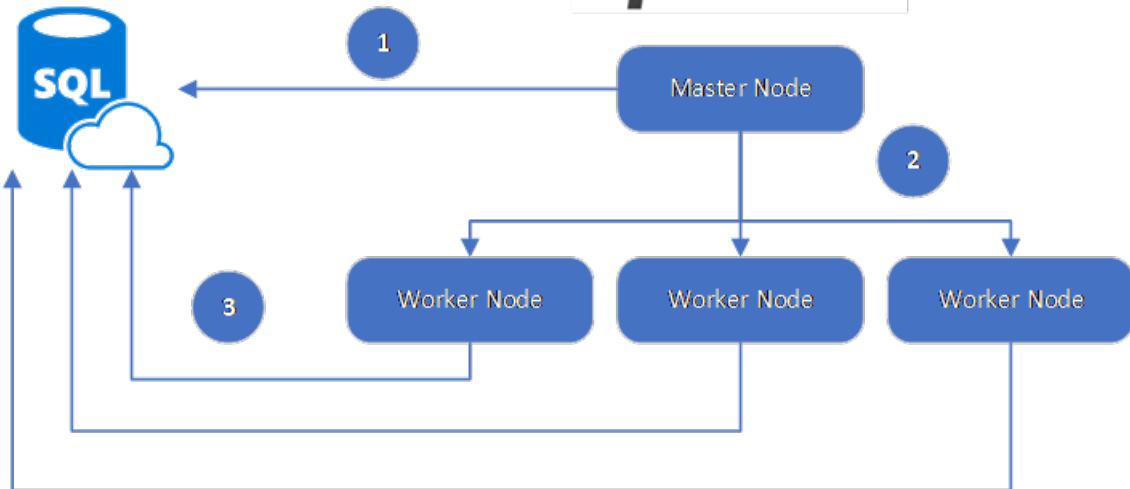
The Spark connector for Azure SQL Database and SQL Server utilizes the Microsoft JDBC Driver for SQL Server to move data between Spark worker nodes and SQL databases:

The dataflow is as follows:

1. The Spark master node connects to SQL Server or Azure SQL Database and loads data from a specific table or using a specific SQL query
2. The Spark master node distributes data to worker nodes for transformation.
3. The Worker node connects to SQL Server or Azure SQL Database and writes data to the database. User can choose to use row-by-row insertion or bulk insert.

The following diagram illustrates the data flow.

## SQL Server or Azure SQL Database



### Build the Spark to SQL DB connector

Currently, the connector project uses maven. To build the connector without dependencies, you can run:

- mvn clean package
- Download the latest versions of the JAR from the release folder
- Include the SQL DB Spark JAR

## Connect Spark to SQL DB using the connector

You can connect to Azure SQL Database or SQL Server from Spark jobs, read or write data. You can also run a DML or DDL query in an Azure SQL database or SQL Server database.

### Read data from Azure SQL Database or SQL Server

```
import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.connect._

val config = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "dbTable" -> "dbo.Clients",
 "user" -> "username",
 "password" -> "*****",
 "connectTimeout" -> "5", //seconds
 "queryTimeout" -> "5" //seconds
))

val collection = sqlContext.read.sqlDB(config)
collection.show()
```

### Reading data from Azure SQL Database or SQL Server with specified SQL query

```

import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.connect._

val config = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "queryCustom" -> "SELECT TOP 100 * FROM dbo.Clients WHERE PostalCode = 98074" //Sql query
 "user" -> "username",
 "password" -> "*****",
))

//Read all data in table dbo.Clients
val collection = sqlContext.read.sqlDB(config)
collection.show()

```

## Write data to Azure SQL Database or SQL Server

```

import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.connect._

// Aquire a DataFrame collection (val collection)

val config = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "dbTable" -> "dbo.Clients",
 "user" -> "username",
 "password" -> "*****"
))

import org.apache.spark.sql.SaveMode
collection.write.mode(SaveMode.Append).sqlDB(config)

```

## Run DML or DDL query in Azure SQL Database or SQL Server

```

import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.query._
val query = """
 |UPDATE Customers
 |SET ContactName = 'Alfred Schmidt', City = 'Frankfurt'
 |WHERE CustomerID = 1;
"""

""".stripMargin

val config = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "user" -> "username",
 "password" -> "*****",
 "queryCustom" -> query
))

sqlContext.sqlDBQuery(config)

```

# Connect Spark to Azure SQL Database using AAD authentication

You can connect to Azure SQL Database using Azure Active Directory (AAD) authentication. Use AAD authentication to centrally manage identities of database users and as an alternative to SQL Server authentication.

### Connecting using ActiveDirectoryPassword Authentication Mode

#### Setup Requirement

If you are using the ActiveDirectoryPassword authentication mode, you need to download [azure-activedirectory-](#)

[library-for-java](#) and its dependencies, and include them in the Java build path.

```
import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.connect._

val config = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "user" -> "username",
 "password" -> "*****",
 "authentication" -> "ActiveDirectoryPassword",
 "encrypt" -> "true"
))

val collection = sqlContext.read.sqlDB(config)
collection.show()
```

## Connecting using Access Token

### Setup Requirement

If you are using the access token-based authentication mode, you need to download [azure-activedirectory-library-for-java](#) and its dependencies, and include them in the Java build path.

See [Use Azure Active Directory Authentication for authentication with SQL Database](#) to learn how to get access token to your Azure SQL database.

```
import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.connect._

val config = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "accessToken" -> "access_token",
 "hostNameInCertificate" -> "*.*.database.windows.net",
 "encrypt" -> "true"
))

val collection = sqlContext.read.sqlDB(config)
collection.show()
```

## Write data to Azure SQL database or SQL Server using Bulk Insert

The traditional jdbc connector writes data into Azure SQL database or SQL Server using row-by-row insertion. You can use Spark to SQL DB connector to write data to SQL database using bulk insert. It significantly improves the write performance when loading large data sets or loading data into tables where a column store index is used.

```

import com.microsoft.azure.sqldb.spark.bulkcopy.BulkCopyMetadata
import com.microsoft.azure.sqldb.spark.config.Config
import com.microsoft.azure.sqldb.spark.connect._

/*
 Add column Metadata.
 If not specified, metadata is automatically added
 from the destination table, which may suffer performance.
*/

var bulkCopyMetadata = new BulkCopyMetadata
bulkCopyMetadata.addColumnMetadata(1, "Title", java.sql.Types.NVARCHAR, 128, 0)
bulkCopyMetadata.addColumnMetadata(2, "FirstName", java.sql.Types.NVARCHAR, 50, 0)
bulkCopyMetadata.addColumnMetadata(3, "LastName", java.sql.Types.NVARCHAR, 50, 0)

val bulkCopyConfig = Config(Map(
 "url" -> "mysqlserver.database.windows.net",
 "databaseName" -> "MyDatabase",
 "user" -> "username",
 "password" -> "*****",
 "dbTable" -> "dbo.Clients",
 "bulkCopyBatchSize" -> "2500",
 "bulkCopyTableLock" -> "true",
 "bulkCopyTimeout" -> "600"
))

df.bulkCopyToSqlDB(bulkCopyConfig, bulkCopyMetadata)
//df.bulkCopyToSqlDB(bulkCopyConfig) if no metadata is specified.

```

## Next steps

If you haven't already, download the Spark connector for Azure SQL Database and SQL Server from [azure-sqldb-spark GitHub repository](#) and explore the additional resources in the repo:

- [Sample Azure Databricks notebooks](#)
- [Sample scripts \(Scala\)](#)

You might also want to review the [Apache Spark SQL, DataFrames, and Datasets Guide](#) and the [Azure Databricks documentation](#).

# Manage historical data in Temporal Tables with retention policy

11/7/2019 • 7 minutes to read • [Edit Online](#)

Temporal Tables may increase database size more than regular tables, especially if you retain historical data for a longer period of time. Hence, retention policy for historical data is an important aspect of planning and managing the lifecycle of every temporal table. Temporal Tables in Azure SQL Database come with easy-to-use retention mechanism that helps you accomplish this task.

Temporal history retention can be configured at the individual table level, which allows users to create flexible aging policies. Applying temporal retention is simple: it requires only one parameter to be set during table creation or schema change.

After you define retention policy, Azure SQL Database starts checking regularly if there are historical rows that are eligible for automatic data cleanup. Identification of matching rows and their removal from the history table occur transparently, in the background task that is scheduled and run by the system. Age condition for the history table rows is checked based on the column representing end of SYSTEM\_TIME period. If retention period, for example, is set to six months, table rows eligible for cleanup satisfy the following condition:

```
ValidTo < DATEADD (MONTH, -6, SYSUTCDATETIME())
```

In the preceding example, we assumed that **ValidTo** column corresponds to the end of SYSTEM\_TIME period.

## How to configure retention policy

Before you configure retention policy for a temporal table, check first whether temporal historical retention is enabled *at the database level*.

```
SELECT is_temporal_history_retention_enabled, name
FROM sys.databases
```

Database flag **is\_temporal\_history\_retention\_enabled** is set to ON by default, but users can change it with ALTER DATABASE statement. It is also automatically set to OFF after [point in time restore](#) operation. To enable temporal history retention cleanup for your database, execute the following statement:

```
ALTER DATABASE <myDB>
SET TEMPORAL_HISTORY_RETENTION ON
```

### IMPORTANT

You can configure retention for temporal tables even if **is\_temporal\_history\_retention\_enabled** is OFF, but automatic cleanup for aged rows is not triggered in that case.

Retention policy is configured during table creation by specifying value for the HISTORY\_RETENTION\_PERIOD parameter:

```

CREATE TABLE dbo.WebsiteUserInfo
(
 [UserID] int NOT NULL PRIMARY KEY CLUSTERED
 , [UserName] nvarchar(100) NOT NULL
 , [PagesVisited] int NOT NULL
 , [ValidFrom] datetime2 (0) GENERATED ALWAYS AS ROW START
 , [ValidTo] datetime2 (0) GENERATED ALWAYS AS ROW END
 , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
)
WITH
(
 SYSTEM_VERSIONING = ON
 (
 HISTORY_TABLE = dbo.WebsiteUserInfoHistory,
 HISTORY_RETENTION_PERIOD = 6 MONTHS
)
);

```

Azure SQL Database allows you to specify retention period by using different time units: DAYS, WEEKS, MONTHS, and YEARS. If HISTORY\_RETENTION\_PERIOD is omitted, INFINITE retention is assumed. You can also use INFINITE keyword explicitly.

In some scenarios, you may want to configure retention after table creation, or to change previously configured value. In that case use ALTER TABLE statement:

```

ALTER TABLE dbo.WebsiteUserInfo
SET (SYSTEM_VERSIONING = ON (HISTORY_RETENTION_PERIOD = 9 MONTHS));

```

#### IMPORTANT

Setting SYSTEM\_VERSIONING to OFF *does not preserve* retention period value. Setting SYSTEM\_VERSIONING to ON without HISTORY\_RETENTION\_PERIOD specified explicitly results in the INFINITE retention period.

To review current state of the retention policy, use the following query that joins temporal retention enablement flag at the database level with retention periods for individual tables:

```

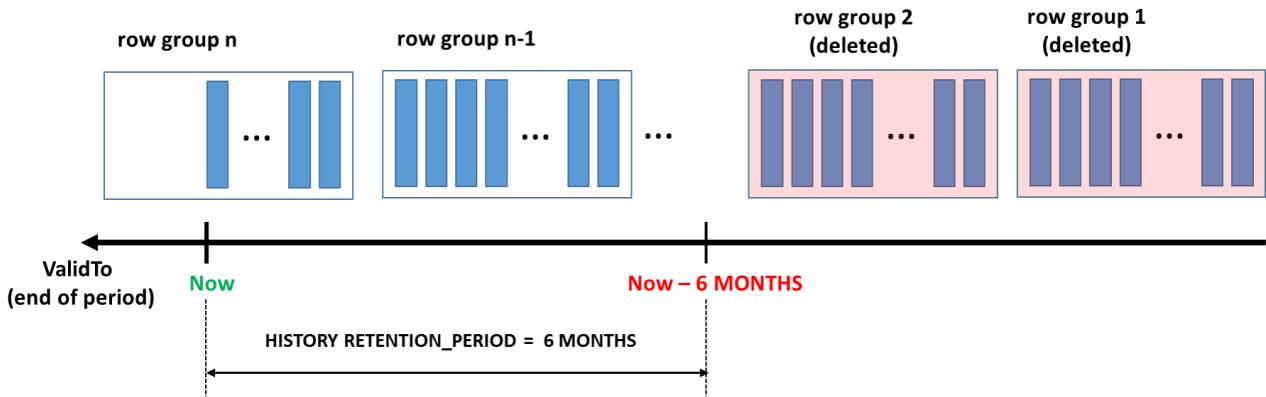
SELECT DB.is_temporal_history_retention_enabled,
SCHEMA_NAME(T1.schema_id) AS TemporalTableSchema,
T1.name as TemporalTableName, SCHEMA_NAME(T2.schema_id) AS HistoryTableSchema,
T2.name as HistoryTableName,T1.history_retention_period,
T1.history_retention_period_unit_desc
FROM sys.tables T1
OUTER APPLY (select is_temporal_history_retention_enabled from sys.databases
where name = DB_NAME()) AS DB
LEFT JOIN sys.tables T2
ON T1.history_table_id = T2.object_id WHERE T1.temporal_type = 2

```

## How SQL Database deletes aged rows

The cleanup process depends on the index layout of the history table. It is important to notice that *only history tables with a clustered index (B-tree or columnstore) can have finite retention policy configured*. A background task is created to perform aged data cleanup for all temporal tables with finite retention period. Cleanup logic for the rowstore (B-tree) clustered index deletes aged row in smaller chunks (up to 10K) minimizing pressure on database log and IO subsystem. Although cleanup logic utilizes required B-tree index, order of deletions for the rows older than retention period cannot be firmly guaranteed. Hence, *do not take any dependency on the cleanup order in your applications*.

The cleanup task for the clustered columnstore removes entire [row groups](#) at once (typically contain 1 million of rows each), which is very efficient, especially when historical data is generated at a high pace.



Excellent data compression and efficient retention cleanup makes clustered columnstore index a perfect choice for scenarios when your workload rapidly generates high amount of historical data. That pattern is typical for intensive [transactional processing workloads that use temporal tables](#) for change tracking and auditing, trend analysis, or IoT data ingestion.

## Index considerations

The cleanup task for tables with rowstore clustered index requires index to start with the column corresponding the end of `SYSTEM_TIME` period. If such index doesn't exist, you cannot configure a finite retention period:

*Msg 13765, Level 16, State 1*

*Setting finite retention period failed on system-versioned temporal table*

*'temporalstagetestdb.dbo.WebsiteUserInfo' because the history table*

*'temporalstagetestdb.dbo.WebsiteUserInfoHistory' does not contain required clustered index. Consider creating a clustered columnstore or B-tree index starting with the column that matches end of `SYSTEM_TIME` period, on the history table.*

It is important to notice that the default history table created by Azure SQL Database already has clustered index, which is compliant for retention policy. If you try to remove that index on a table with finite retention period, operation fails with the following error:

*Msg 13766, Level 16, State 1*

*Cannot drop the clustered index 'WebsiteUserInfoHistory.IX\_WebsiteUserInfoHistory' because it is being used for automatic cleanup of aged data. Consider setting `HISTORY_RETENTION_PERIOD` to `INFINITE` on the corresponding system-versioned temporal table if you need to drop this index.*

Cleanup on the clustered columnstore index works optimally if historical rows are inserted in the ascending order (ordered by the end of period column), which is always the case when the history table is populated exclusively by the `SYSTEM_VERSIONING` mechanism. If rows in the history table are not ordered by end of period column (which may be the case if you migrated existing historical data), you should re-create clustered columnstore index on top of B-tree rowstore index that is properly ordered, to achieve optimal performance.

Avoid rebuilding clustered columnstore index on the history table with the finite retention period, because it may change ordering in the row groups naturally imposed by the system-versioning operation. If you need to rebuild clustered columnstore index on the history table, do that by re-creating it on top of compliant B-tree index, preserving ordering in the rowgroups necessary for regular data cleanup. The same approach should be taken if you create temporal table with existing history table that has clustered column index without guaranteed data order:

```
/*Create B-tree ordered by the end of period column*/
CREATE CLUSTERED INDEX IX_WebsiteUserInfoHistory ON WebsiteUserInfoHistory (ValidTo)
WITH (DROP_EXISTING = ON);
GO
/*Re-create clustered columnstore index*/
CREATE CLUSTERED COLUMNSTORE INDEX IX_WebsiteUserInfoHistory ON WebsiteUserInfoHistory
WITH (DROP_EXISTING = ON);
```

When finite retention period is configured for the history table with the clustered columnstore index, you cannot create additional non-clustered B-tree indexes on that table:

```
CREATE NONCLUSTERED INDEX IX_WebHistNCI ON WebsiteUserInfoHistory ([UserName])
```

An attempt to execute above statement fails with the following error:

*Msg 13772, Level 16, State 1*

*Cannot create non-clustered index on a temporal history table 'WebsiteUserInfoHistory' since it has finite retention period and clustered columnstore index defined.*

## Querying tables with retention policy

All queries on the temporal table automatically filter out historical rows matching finite retention policy, to avoid unpredictable and inconsistent results, since aged rows can be deleted by the cleanup task, *at any point in time and in arbitrary order*.

The following picture shows the query plan for a simple query:

```
SELECT * FROM dbo.WebsiteUserInfo FOR SYSTEM_TIME ALL;
```

The query plan includes additional filter applied to end of period column (ValidTo) in the Clustered Index Scan operator on the history table (highlighted). This example assumes that one MONTH retention period was set on WebsiteUserInfo table.

Object Explorer

SQLQuery6.sql - sql...stdb (BaneSa (116)) \* X SQLQuery2.sql - (l...UROPE\bonova (55)) \* SQLQuery3.sql - sql...stdb (BaneSa (121)) \* Query

```
SELECT * FROM WebsiteUserInfo
FOR SYSTEM_TIME ALL
```

100 %

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT \* FROM WebsiteUserInfo FOR SYSTEM\_TIME ALL

**Physical Operation** Clustered Index Scan  
**Logical Operation** Clustered Index Scan  
**Actual Execution Mode** Row  
**Estimated Execution Mode** Row  
**Storage** RowStore  
**Actual Number of Rows** 0  
**Actual Number of Batches** 0  
**Estimated Operator Cost** 0.0032831 (50%)  
**Estimated I/O Cost** 0.003125  
**Estimated CPU Cost** 0.0001581  
**Estimated Subtree Cost** 0.0032831  
**Number of Executions** 1  
**Estimated Number of Executions** 1  
**Estimated Number of Rows** 1  
**Estimated Row Size** 131 B  
**Actual Rebinds** 0  
**Actual Rewinds** 0  
**Ordered** False  
**Node ID** 2

**Predicate**

```
[temporalstagetestdb].[dbo].[WebsiteUserInfoHistory]. [ValidTo] >= dateadd(month, (-1), sysutcdatetime()) AND [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory]. [ValidFrom] <> [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory]. [ValidTo]
```

**Object**

```
[temporalstagetestdb].[dbo].[WebsiteUserInfoHistory]. [IX_WebsiteUserInfoHistory]
```

**Output List**

```
[temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].UserID, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].UserName, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].PagesVisited, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].ValidFrom, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].ValidTo
```

Query executed successfully.

Find Results 1

Ready

However, if you query history table directly, you may see rows that are older than specified retention period, but without any guarantee for repeatable query results. The following picture shows query execution plan for the query on the history table without additional filters applied:

SQLQuery9.sql - sq...STDB (BaneSa (121))\* X SQLQuery8.sql - not connected\* SQLQuery7.sql - not

```
SELECT * FROM dbo.WebsiteUserInfoHistory
```

100 %

Results Messages Execution plan

Query 1: Query cost (relative)

```
SELECT * FROM dbo.WebsiteUser
```

SELECT Cost: 0

Clustered Index Scan [WebsiteUserInfoHistory]

Cost: 1

**Physical Operation**

Logical Operation Clustered Index Scan

Actual Execution Mode Row

Estimated Execution Mode Row

Storage RowStore

Actual Number of Rows 0

Actual Number of Batches 0

Estimated I/O Cost 0.003125

Estimated Operator Cost 0.0032831 (100%)

Estimated CPU Cost 0.0001581

Estimated Subtree Cost 0.0032831

Number of Executions 1

Estimated Number of Executions 1

Estimated Number of Rows 1

Estimated Row Size 131 B

Actual Rebinds 0

Actual Rewinds 0

Ordered False

Node ID 0

**Object**

[temporalstagetestdb].[dbo].[WebsiteUserInfoHistory]. [IX\_WebsiteUserInfoHistory]

**Output List**

[temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].UserID, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].UserName, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].PagesVisited, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].ValidFrom, [temporalstagetestdb].[dbo].[WebsiteUserInfoHistory].ValidTo

Query executed successfully.

Find Results 1

Ready

Do not rely your business logic on reading history table beyond retention period as you may get inconsistent or unexpected results. We recommend that you use temporal queries with FOR SYSTEM\_TIME clause for analyzing data in temporal tables.

## Point in time restore considerations

When you create new database by [restoring existing database to a specific point in time](#), it has temporal retention disabled at the database level. (**is\_temporal\_history\_retention\_enabled** flag set to OFF). This functionality allows you to examine all historical rows upon restore, without worrying that aged rows are removed before you get to query them. You can use it to *inspect historical data beyond configured retention period*.

Say that a temporal table has one MONTH retention period specified. If your database was created in Premium Service tier, you would be able to create database copy with the database state up to 35 days back in the past. That effectively would allow you to analyze historical rows that are up to 65 days old by querying the history table directly.

If you want to activate temporal retention cleanup, run the following Transact-SQL statement after point in time restore:

```
ALTER DATABASE <myDB>
SET TEMPORAL_HISTORY_RETENTION ON
```

## Next steps

To learn how to use Temporal Tables in your applications, check out [Getting Started with Temporal Tables in Azure SQL Database](#).

Visit Channel 9 to hear a [real customer temporal implementation success story](#) and watch a [live temporal demonstration](#).

For detailed information about Temporal Tables, review [MSDN documentation](#).

# Get the required values for authenticating an application to access SQL Database from code

11/22/2019 • 2 minutes to read • [Edit Online](#)

To create and manage SQL Database from code you must register your app in the Azure Active Directory (AAD) domain in the subscription where your Azure resources have been created.

## Create a service principal to access resources from an application

The following examples create the Active Directory (AD) application and the service principal that we need to authenticate our C# app. The script outputs values we need for the preceding C# sample. For detailed information, see [Use Azure PowerShell to create a service principal to access resources](#).

- [PowerShell](#)
- [Azure CLI](#)

### IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

```
sign in to Azure
Connect-AzAccount

for multiple subscriptions, uncomment and set to the subscription you want to work with
#$subscriptionId = "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}"
#Set-AzContext -SubscriptionId $subscriptionId

$appName = "{app-name}" # display name for your app, must be unique in your directory
$uri = "http://{app-name}" # does not need to be a real uri
$secret = "{app-password}"

create an AAD app
$azureAdApplication = New-AzADApplication -DisplayName $appName -HomePage $Uri -IdentifierUris $Uri -Password
$secret

create a Service Principal for the app
$svcprincipal = New-AzADServicePrincipal -ApplicationId $azureAdApplication.ApplicationId

Start-Sleep -s 15 # to avoid a PrincipalNotFound error, pause here for 15 seconds

if you still get a PrincipalNotFound error, then rerun the following until successful.
$roleassignment = New-AzRoleAssignment -RoleDefinitionName Contributor -ServicePrincipalName
$azureAdApplication.ApplicationId.Guid

output the values we need for our C# application to successfully authenticate
Write-Output "Copy these values into the C# sample app"

Write-Output "_subscriptionId:" (Get-AzContext).Subscription.SubscriptionId
Write-Output "_tenantId:" (Get-AzContext).Tenant.TenantId
Write-Output "_applicationId:" $azureAdApplication.ApplicationId.Guid
Write-Output "_applicationSecret:" $secret
```

## See also

[Create a SQL database with C#](#)

[Connecting to SQL Database By Using Azure Active Directory Authentication](#)

# How to use batching to improve SQL Database application performance

1/23/2020 • 25 minutes to read • [Edit Online](#)

Batching operations to Azure SQL Database significantly improves the performance and scalability of your applications. In order to understand the benefits, the first part of this article covers some sample test results that compare sequential and batched requests to a SQL Database. The remainder of the article shows the techniques, scenarios, and considerations to help you to use batching successfully in your Azure applications.

## Why is batching important for SQL Database

Batching calls to a remote service is a well-known strategy for increasing performance and scalability. There are fixed processing costs to any interactions with a remote service, such as serialization, network transfer, and deserialization. Packaging many separate transactions into a single batch minimizes these costs.

In this paper, we want to examine various SQL Database batching strategies and scenarios. Although these strategies are also important for on-premises applications that use SQL Server, there are several reasons for highlighting the use of batching for SQL Database:

- There is potentially greater network latency in accessing SQL Database, especially if you are accessing SQL Database from outside the same Microsoft Azure datacenter.
- The multitenant characteristics of SQL Database means that the efficiency of the data access layer correlates to the overall scalability of the database. SQL Database must prevent any single tenant/user from monopolizing database resources to the detriment of other tenants. In response to usage in excess of predefined quotas, SQL Database can reduce throughput or respond with throttling exceptions. Efficiencies, such as batching, enable you to do more work on SQL Database before reaching these limits.
- Batching is also effective for architectures that use multiple databases (sharding). The efficiency of your interaction with each database unit is still a key factor in your overall scalability.

One of the benefits of using SQL Database is that you don't have to manage the servers that host the database. However, this managed infrastructure also means that you have to think differently about database optimizations. You can no longer look to improve the database hardware or network infrastructure. Microsoft Azure controls those environments. The main area that you can control is how your application interacts with SQL Database. Batching is one of these optimizations.

The first part of the paper examines various batching techniques for .NET applications that use SQL Database. The last two sections cover batching guidelines and scenarios.

## Batching strategies

### Note about timing results in this article

#### NOTE

Results are not benchmarks but are meant to show **relative performance**. Timings are based on an average of at least 10 test runs. Operations are inserts into an empty table. These tests were measured pre-V12, and they do not necessarily correspond to throughput that you might experience in a V12 database using the new [DTU service tiers](#) or [vCore service tiers](#). The relative benefit of the batching technique should be similar.

### Transactions

It seems strange to begin a review of batching by discussing transactions. But the use of client-side transactions has a subtle server-side batching effect that improves performance. And transactions can be added with only a few lines of code, so they provide a fast way to improve performance of sequential operations.

Consider the following C# code that contains a sequence of insert and update operations on a simple table.

```
List<string> dbOperations = new List<string>();
dbOperations.Add("update MyTable set mytext = 'updated text' where id = 1");
dbOperations.Add("update MyTable set mytext = 'updated text' where id = 2");
dbOperations.Add("update MyTable set mytext = 'updated text' where id = 3");
dbOperations.Add("insert MyTable values ('new value',1)");
dbOperations.Add("insert MyTable values ('new value',2)");
dbOperations.Add("insert MyTable values ('new value',3)");
```

The following ADO.NET code sequentially performs these operations.

```
using (SqlConnection connection = new
SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))
{
 conn.Open();

 foreach(string commandString in dbOperations)
 {
 SqlCommand cmd = new SqlCommand(commandString, conn);
 cmd.ExecuteNonQuery();
 }
}
```

The best way to optimize this code is to implement some form of client-side batching of these calls. But there is a simple way to increase the performance of this code by simply wrapping the sequence of calls in a transaction. Here is the same code that uses a transaction.

```
using (SqlConnection connection = new
SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))
{
 conn.Open();
 SqlTransaction transaction = conn.BeginTransaction();

 foreach (string commandString in dbOperations)
 {
 SqlCommand cmd = new SqlCommand(commandString, conn, transaction);
 cmd.ExecuteNonQuery();
 }

 transaction.Commit();
}
```

Transactions are actually being used in both of these examples. In the first example, each individual call is an implicit transaction. In the second example, an explicit transaction wraps all of the calls. Per the documentation for the [write-ahead transaction log](#), log records are flushed to the disk when the transaction commits. So by including more calls in a transaction, the write to the transaction log can delay until the transaction is committed. In effect, you are enabling batching for the writes to the server's transaction log.

The following table shows some ad hoc testing results. The tests performed the same sequential inserts with and without transactions. For more perspective, the first set of tests ran remotely from a laptop to the database in Microsoft Azure. The second set of tests ran from a cloud service and database that both resided within the same Microsoft Azure datacenter (West US). The following table shows the duration in milliseconds of sequential inserts with and without transactions.

## On-Premises to Azure:

OPERATIONS	NO TRANSACTION (MS)	TRANSACTION (MS)
1	130	402
10	1208	1226
100	12662	10395
1000	128852	102917

## Azure to Azure (same datacenter):

OPERATIONS	NO TRANSACTION (MS)	TRANSACTION (MS)
1	21	26
10	220	56
100	2145	341
1000	21479	2756

### NOTE

Results are not benchmarks. See the [note about timing results in this article](#).

Based on the previous test results, wrapping a single operation in a transaction actually decreases performance. But as you increase the number of operations within a single transaction, the performance improvement becomes more marked. The performance difference is also more noticeable when all operations occur within the Microsoft Azure datacenter. The increased latency of using SQL Database from outside the Microsoft Azure datacenter overshadows the performance gain of using transactions.

Although the use of transactions can increase performance, continue to [observe best practices for transactions and connections](#). Keep the transaction as short as possible, and close the database connection after the work completes. The using statement in the previous example assures that the connection is closed when the subsequent code block completes.

The previous example demonstrates that you can add a local transaction to any ADO.NET code with two lines. Transactions offer a quick way to improve the performance of code that makes sequential insert, update, and delete operations. However, for the fastest performance, consider changing the code further to take advantage of client-side batching, such as table-valued parameters.

For more information about transactions in ADO.NET, see [Local Transactions in ADO.NET](#).

## Table-valued parameters

Table-valued parameters support user-defined table types as parameters in Transact-SQL statements, stored procedures, and functions. This client-side batching technique allows you to send multiple rows of data within the table-valued parameter. To use table-valued parameters, first define a table type. The following Transact-SQL statement creates a table type named **MyTableType**.

```

CREATE TYPE MyTableType AS TABLE
(mytext TEXT,
 num INT);

```

In code, you create a **DataTable** with the exact same names and types of the table type. Pass this **DataTable** in a parameter in a text query or stored procedure call. The following example shows this technique:

```

using (SqlConnection connection = new
SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))
{
 connection.Open();

 DataTable table = new DataTable();
 // Add columns and rows. The following is a simple example.
 table.Columns.Add("mytext", typeof(string));
 table.Columns.Add("num", typeof(int));
 for (var i = 0; i < 10; i++)
 {
 table.Rows.Add(DateTime.Now.ToString(), DateTime.Now.Millisecond);
 }

 SqlCommand cmd = new SqlCommand(
 "INSERT INTO MyTable(mytext, num) SELECT mytext, num FROM @TestTvp",
 connection);

 cmd.Parameters.Add(
 new SqlParameter()
 {
 ParameterName = "@TestTvp",
 SqlDbType = SqlDbType.Structured,
 TypeName = "MyTableType",
 Value = table,
 });

 cmd.ExecuteNonQuery();
}

```

In the previous example, the **SqlCommand** object inserts rows from a table-valued parameter, **@TestTvp**. The previously created **DataTable** object is assigned to this parameter with the **SqlCommand.Parameters.Add** method. Batching the inserts in one call significantly increases the performance over sequential inserts.

To improve the previous example further, use a stored procedure instead of a text-based command. The following Transact-SQL command creates a stored procedure that takes the **SimpleTestTableType** table-valued parameter.

```

CREATE PROCEDURE [dbo].[sp_InsertRows]
@TestTvp AS MyTableType READONLY
AS
BEGIN
 INSERT INTO MyTable(mytext, num)
 SELECT mytext, num FROM @TestTvp
END
GO

```

Then change the **SqlCommand** object declaration in the previous code example to the following.

```

SqlCommand cmd = new SqlCommand("sp_InsertRows", connection);
cmd.CommandType = CommandType.StoredProcedure;

```

In most cases, table-valued parameters have equivalent or better performance than other batching techniques.

Table-valued parameters are often preferable, because they are more flexible than other options. For example, other techniques, such as SQL bulk copy, only permit the insertion of new rows. But with table-valued parameters, you can use logic in the stored procedure to determine which rows are updates and which are inserts. The table type can also be modified to contain an "Operation" column that indicates whether the specified row should be inserted, updated, or deleted.

The following table shows ad hoc test results for the use of table-valued parameters in milliseconds.

OPERATIONS	ON-PREMISES TO AZURE (MS)	AZURE SAME DATACENTER (MS)
1	124	32
10	131	25
100	338	51
1000	2615	382
10000	23830	3586

#### NOTE

Results are not benchmarks. See the [note about timing results in this article](#).

The performance gain from batching is immediately apparent. In the previous sequential test, 1000 operations took 129 seconds outside the datacenter and 21 seconds from within the datacenter. But with table-valued parameters, 1000 operations take only 2.6 seconds outside the datacenter and 0.4 seconds within the datacenter.

For more information on table-valued parameters, see [Table-Valued Parameters](#).

#### SQL bulk copy

SQL bulk copy is another way to insert large amounts of data into a target database. .NET applications can use the **SqlBulkCopy** class to perform bulk insert operations. **SqlBulkCopy** is similar in function to the command-line tool, **Bcp.exe**, or the Transact-SQL statement, **BULK INSERT**. The following code example shows how to bulk copy the rows in the source **DataTable**, table, to the destination table in SQL Server, MyTable.

```
using (SqlConnection connection = new
SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))
{
 connection.Open();

 using (SqlBulkCopy bulkCopy = new SqlBulkCopy(connection))
 {
 bulkCopy.DestinationTableName = "MyTable";
 bulkCopy.ColumnMappings.Add("mytext", "mytext");
 bulkCopy.ColumnMappings.Add("num", "num");
 bulkCopy.WriteToServer(table);
 }
}
```

There are some cases where bulk copy is preferred over table-valued parameters. See the comparison table of Table-Valued parameters versus BULK INSERT operations in the article [Table-Valued Parameters](#).

The following ad hoc test results show the performance of batching with **SqlBulkCopy** in milliseconds.

OPERATIONS	ON-PREMISES TO AZURE (MS)	AZURE SAME DATACENTER (MS)
1	433	57
10	441	32
100	636	53
1000	2535	341
10000	21605	2737

#### NOTE

Results are not benchmarks. See the [note about timing results in this article](#).

In smaller batch sizes, the use table-valued parameters outperformed the **SqlBulkCopy** class. However, **SqlBulkCopy** performed 12-31% faster than table-valued parameters for the tests of 1,000 and 10,000 rows. Like table-valued parameters, **SqlBulkCopy** is a good option for batched inserts, especially when compared to the performance of non-batched operations.

For more information on bulk copy in ADO.NET, see [Bulk Copy Operations in SQL Server](#).

#### Multiple-row Parameterized INSERT statements

One alternative for small batches is to construct a large parameterized INSERT statement that inserts multiple rows. The following code example demonstrates this technique.

```
using (SqlConnection connection = new
SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))
{
 connection.Open();

 string insertCommand = "INSERT INTO [MyTable] (mytext, num) " +
 "VALUES (@p1, @p2), (@p3, @p4), (@p5, @p6), (@p7, @p8), (@p9, @p10)";

 SqlCommand cmd = new SqlCommand(insertCommand, connection);

 for (int i = 1; i <= 10; i += 2)
 {
 cmd.Parameters.Add(new SqlParameter("@p" + i.ToString(), "test"));
 cmd.Parameters.Add(new SqlParameter("@p" + (i+1).ToString(), i));
 }

 cmd.ExecuteNonQuery();
}
```

This example is meant to show the basic concept. A more realistic scenario would loop through the required entities to construct the query string and the command parameters simultaneously. You are limited to a total of 2100 query parameters, so this limits the total number of rows that can be processed in this manner.

The following ad hoc test results show the performance of this type of insert statement in milliseconds.

OPERATIONS	TABLE-VALUED PARAMETERS (MS)	SINGLE-STATEMENT INSERT (MS)
1	32	20

OPERATIONS	TABLE-VALUED PARAMETERS (MS)	SINGLE-STATEMENT INSERT (MS)
10	30	25
100	33	51

#### NOTE

Results are not benchmarks. See the [note about timing results in this article](#).

This approach can be slightly faster for batches that are less than 100 rows. Although the improvement is small, this technique is another option that might work well in your specific application scenario.

#### DataAdapter

The **DataAdapter** class allows you to modify a **DataSet** object and then submit the changes as INSERT, UPDATE, and DELETE operations. If you are using the **DataAdapter** in this manner, it is important to note that separate calls are made for each distinct operation. To improve performance, use the **UpdateBatchSize** property to the number of operations that should be batched at a time. For more information, see [Performing Batch Operations Using DataAdapters](#).

#### Entity framework

Entity Framework does not currently support batching. Different developers in the community have attempted to demonstrate workarounds, such as override the **SaveChanges** method. But the solutions are typically complex and customized to the application and data model. The Entity Framework codeplex project currently has a discussion page on this feature request. To view this discussion, see [Design Meeting Notes - August 2, 2012](#).

#### XML

For completeness, we feel that it is important to talk about XML as a batching strategy. However, the use of XML has no advantages over other methods and several disadvantages. The approach is similar to table-valued parameters, but an XML file or string is passed to a stored procedure instead of a user-defined table. The stored procedure parses the commands in the stored procedure.

There are several disadvantages to this approach:

- Working with XML can be cumbersome and error prone.
- Parsing the XML on the database can be CPU-intensive.
- In most cases, this method is slower than table-valued parameters.

For these reasons, the use of XML for batch queries is not recommended.

## Batching considerations

The following sections provide more guidance for the use of batching in SQL Database applications.

#### Tradeoffs

Depending on your architecture, batching can involve a tradeoff between performance and resiliency. For example, consider the scenario where your role unexpectedly goes down. If you lose one row of data, the impact is smaller than the impact of losing a large batch of unsubmitted rows. There is a greater risk when you buffer rows before sending them to the database in a specified time window.

Because of this tradeoff, evaluate the type of operations that you batch. Batch more aggressively (larger batches and longer time windows) with data that is less critical.

#### Batch size

In our tests, there was typically no advantage to breaking large batches into smaller chunks. In fact, this subdivision often resulted in slower performance than submitting a single large batch. For example, consider a scenario where you want to insert 1000 rows. The following table shows how long it takes to use table-valued parameters to insert 1000 rows when divided into smaller batches.

BATCH SIZE	ITERATIONS	TABLE-VALUED PARAMETERS (MS)
1000	1	347
500	2	355
100	10	465
50	20	630

#### NOTE

Results are not benchmarks. See the [note about timing results in this article](#).

You can see that the best performance for 1000 rows is to submit them all at once. In other tests (not shown here), there was a small performance gain to break a 10000 row batch into two batches of 5000. But the table schema for these tests is relatively simple, so you should perform tests on your specific data and batch sizes to verify these findings.

Another factor to consider is that if the total batch becomes too large, SQL Database might throttle and refuse to commit the batch. For the best results, test your specific scenario to determine if there is an ideal batch size. Make the batch size configurable at runtime to enable quick adjustments based on performance or errors.

Finally, balance the size of the batch with the risks associated with batching. If there are transient errors or the role fails, consider the consequences of retrying the operation or of losing the data in the batch.

#### Parallel processing

What if you took the approach of reducing the batch size but used multiple threads to execute the work? Again, our tests showed that several smaller multithreaded batches typically performed worse than a single larger batch. The following test attempts to insert 1000 rows in one or more parallel batches. This test shows how more simultaneous batches actually decreased performance.

BATCH SIZE [ITERATIONS]	TWO THREADS (MS)	FOUR THREADS (MS)	SIX THREADS (MS)
1000 [1]	277	315	266
500 [2]	548	278	256
250 [4]	405	329	265
100 [10]	488	439	391

#### NOTE

Results are not benchmarks. See the [note about timing results in this article](#).

There are several potential reasons for the degradation in performance due to parallelism:

- There are multiple simultaneous network calls instead of one.
- Multiple operations against a single table can result in contention and blocking.
- There are overheads associated with multithreading.
- The expense of opening multiple connections outweighs the benefit of parallel processing.

If you target different tables or databases, it is possible to see some performance gain with this strategy. Database sharding or federations would be a scenario for this approach. Sharding uses multiple databases and routes different data to each database. If each small batch is going to a different database, then performing the operations in parallel can be more efficient. However, the performance gain is not significant enough to use as the basis for a decision to use database sharding in your solution.

In some designs, parallel execution of smaller batches can result in improved throughput of requests in a system under load. In this case, even though it is quicker to process a single larger batch, processing multiple batches in parallel might be more efficient.

If you do use parallel execution, consider controlling the maximum number of worker threads. A smaller number might result in less contention and a faster execution time. Also, consider the additional load that this places on the target database both in connections and transactions.

### **Related performance factors**

Typical guidance on database performance also affects batching. For example, insert performance is reduced for tables that have a large primary key or many nonclustered indexes.

If table-valued parameters use a stored procedure, you can use the command **SET NOCOUNT ON** at the beginning of the procedure. This statement suppresses the return of the count of the affected rows in the procedure. However, in our tests, the use of **SET NOCOUNT ON** either had no effect or decreased performance. The test stored procedure was simple with a single **INSERT** command from the table-valued parameter. It is possible that more complex stored procedures would benefit from this statement. But don't assume that adding **SET NOCOUNT ON** to your stored procedure automatically improves performance. To understand the effect, test your stored procedure with and without the **SET NOCOUNT ON** statement.

## **Batching scenarios**

The following sections describe how to use table-valued parameters in three application scenarios. The first scenario shows how buffering and batching can work together. The second scenario improves performance by performing master-detail operations in a single stored procedure call. The final scenario shows how to use table-valued parameters in an "UPSERT" operation.

### **Buffering**

Although there are some scenarios that are obvious candidate for batching, there are many scenarios that could take advantage of batching by delayed processing. However, delayed processing also carries a greater risk that the data is lost in the event of an unexpected failure. It is important to understand this risk and consider the consequences.

For example, consider a web application that tracks the navigation history of each user. On each page request, the application could make a database call to record the user's page view. But higher performance and scalability can be achieved by buffering the users' navigation activities and then sending this data to the database in batches. You can trigger the database update by elapsed time and/or buffer size. For example, a rule could specify that the batch should be processed after 20 seconds or when the buffer reaches 1000 items.

The following code example uses [Reactive Extensions - Rx](#) to process buffered events raised by a monitoring class. When the buffer fills or a timeout is reached, the batch of user data is sent to the database with a table-valued parameter.

The following `NavHistoryData` class models the user navigation details. It contains basic information such as the

user identifier, the URL accessed, and the access time.

```
public class NavHistoryData
{
 public NavHistoryData(int userId, string url, DateTime accessTime)
 { UserId = userId; URL = url; AccessTime = accessTime; }
 public int UserId { get; set; }
 public string URL { get; set; }
 public DateTime AccessTime { get; set; }
}
```

The NavHistoryDataMonitor class is responsible for buffering the user navigation data to the database. It contains a method, RecordUserNavigationEntry, which responds by raising an **OnAdded** event. The following code shows the constructor logic that uses Rx to create an observable collection based on the event. It then subscribes to this observable collection with the Buffer method. The overload specifies that the buffer should be sent every 20 seconds or 1000 entries.

```
public NavHistoryDataMonitor()
{
 var observableData =
 Observable.FromEventPattern<NavHistoryDataEventArgs>(this, "OnAdded");

 observableData.Buffer(TimeSpan.FromSeconds(20), 1000).Subscribe(Handler);
}
```

The handler converts all of the buffered items into a table-valued type and then passes this type to a stored procedure that processes the batch. The following code shows the complete definition for both the NavHistoryDataEventArgs and the NavHistoryDataMonitor classes.

```
public class NavHistoryDataEventArgs : System.EventArgs
{
 public NavHistoryDataEventArgs(NavHistoryData data) { Data = data; }
 public NavHistoryData Data { get; set; }
}

public class NavHistoryDataMonitor
{
 public event EventHandler<NavHistoryDataEventArgs> OnAdded;

 public NavHistoryDataMonitor()
 {
 var observableData =
 Observable.FromEventPattern<NavHistoryDataEventArgs>(this, "OnAdded");

 observableData.Buffer(TimeSpan.FromSeconds(20), 1000).Subscribe(Handler);
 }
}
```

The handler converts all of the buffered items into a table-valued type and then passes this type to a stored procedure that processes the batch. The following code shows the complete definition for both the NavHistoryDataEventArgs and the NavHistoryDataMonitor classes.

```

public class NavHistoryDataEventArgs : System.EventArgs
{
 if (OnAdded != null)
 OnAdded(this, new NavHistoryDataEventArgs(data));
}

protected void Handler(IList<EventPattern<NavHistoryDataEventArgs>> items)
{
 DataTable navHistoryBatch = new DataTable("NavigationHistoryBatch");
 navHistoryBatch.Columns.Add("UserId", typeof(int));
 navHistoryBatch.Columns.Add("URL", typeof(string));
 navHistoryBatch.Columns.Add("AccessTime", typeof(DateTime));
 foreach (EventPattern<NavHistoryDataEventArgs> item in items)
 {
 NavHistoryData data = item.EventArgs.Data;
 navHistoryBatch.Rows.Add(data.UserId, data.URL, data.AccessTime);
 }

 using (SqlConnection connection = new
SqlConnection(CloudConfigurationManager.GetSetting("Sql.ConnectionString")))
 {
 connection.Open();

 SqlCommand cmd = new SqlCommand("sp_RecordUserNavigation", connection);
 cmd.CommandType = CommandType.StoredProcedure;

 cmd.Parameters.Add(
 new SqlParameter()
 {
 ParameterName = "@NavHistoryBatch",
 SqlDbType = SqlDbType.Structured,
 TypeName = "NavigationHistoryTableType",
 Value = navHistoryBatch,
 });
 cmd.ExecuteNonQuery();
 }
}
}

```

To use this buffering class, the application creates a static `NavHistoryDataMonitor` object. Each time a user accesses a page, the application calls the `NavHistoryDataMonitor.RecordUserNavigationEntry` method. The buffering logic proceeds to take care of sending these entries to the database in batches.

### **Master detail**

Table-valued parameters are useful for simple INSERT scenarios. However, it can be more challenging to batch inserts that involve more than one table. The “master/detail” scenario is a good example. The master table identifies the primary entity. One or more detail tables store more data about the entity. In this scenario, foreign key relationships enforce the relationship of details to a unique master entity. Consider a simplified version of a `PurchaseOrder` table and its associated `OrderDetail` table. The following Transact-SQL creates the `PurchaseOrder` table with four columns: `OrderID`, `OrderDate`, `CustomerID`, and `Status`.

```

CREATE TABLE [dbo].[PurchaseOrder](
[OrderID] [int] IDENTITY(1,1) NOT NULL,
[OrderDate] [datetime] NOT NULL,
[CustomerID] [int] NOT NULL,
[Status] [nvarchar](50) NOT NULL,
CONSTRAINT [PrimaryKey_PurchaseOrder]
PRIMARY KEY CLUSTERED ([OrderID] ASC)

```

Each order contains one or more product purchases. This information is captured in the `PurchaseOrderDetail`

table. The following Transact-SQL creates the PurchaseOrderDetail table with five columns: OrderID, OrderDetailID, ProductID, UnitPrice, and OrderQty.

```
CREATE TABLE [dbo].[PurchaseOrderDetail](
[OrderID] [int] NOT NULL,
[OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
[ProductID] [int] NOT NULL,
[UnitPrice] [money] NULL,
[OrderQty] [smallint] NULL,
CONSTRAINT [PrimaryKey_PurchaseOrderDetail] PRIMARY KEY CLUSTERED
([OrderID] ASC, [OrderDetailID] ASC))
```

The OrderID column in the PurchaseOrderDetail table must reference an order from the PurchaseOrder table. The following definition of a foreign key enforces this constraint.

```
ALTER TABLE [dbo].[PurchaseOrderDetail] WITH CHECK ADD
CONSTRAINT [FK_OrderID_PurchaseOrder] FOREIGN KEY([OrderID])
REFERENCES [dbo].[PurchaseOrder] ([OrderID])
```

In order to use table-valued parameters, you must have one user-defined table type for each target table.

```
CREATE TYPE PurchaseOrderTableType AS TABLE
(OrderID INT,
 OrderDate DATETIME,
 CustomerID INT,
 Status NVARCHAR(50));
GO

CREATE TYPE PurchaseOrderDetailTableType AS TABLE
(OrderID INT,
 ProductID INT,
 UnitPrice MONEY,
 OrderQty SMALLINT);
GO
```

Then define a stored procedure that accepts tables of these types. This procedure allows an application to locally batch a set of orders and order details in a single call. The following Transact-SQL provides the complete stored procedure declaration for this purchase order example.

```

CREATE PROCEDURE sp_InsertOrdersBatch (
 @orders as PurchaseOrderTableType READONLY,
 @details as PurchaseOrderDetailTableType READONLY)
AS
SET NOCOUNT ON;

-- Table that connects the order identifiers in the @orders
-- table with the actual order identifiers in the PurchaseOrder table
DECLARE @IdentityLink AS TABLE (
 SubmittedKey int,
 ActualKey int,
 RowNumber int identity(1,1)
);

-- Add new orders to the PurchaseOrder table, storing the actual
-- order identifiers in the @IdentityLink table
INSERT INTO PurchaseOrder ([OrderDate], [CustomerID], [Status])
OUTPUT inserted.OrderID INTO @IdentityLink (ActualKey)
SELECT [OrderDate], [CustomerID], [Status] FROM @orders ORDER BY OrderID;

-- Match the passed-in order identifiers with the actual identifiers
-- and complete the @IdentityLink table for use with inserting the details
WITH OrderedRows As (
 SELECT OrderID, ROW_NUMBER () OVER (ORDER BY OrderID) As RowNumber
 FROM @orders
)
UPDATE @IdentityLink SET SubmittedKey = M.OrderID
FROM @IdentityLink L JOIN OrderedRows M ON L.RowNumber = M.RowNumber;

-- Insert the order details into the PurchaseOrderDetail table,
-- using the actual order identifiers of the master table, PurchaseOrder
INSERT INTO PurchaseOrderDetail (
 [OrderID],
 [ProductID],
 [UnitPrice],
 [OrderQty])
SELECT L.ActualKey, D.ProductID, D.UnitPrice, D.OrderQty
FROM @details D
JOIN @IdentityLink L ON L.SubmittedKey = D.OrderID;
GO

```

In this example, the locally defined @IdentityLink table stores the actual OrderID values from the newly inserted rows. These order identifiers are different from the temporary OrderID values in the @orders and @details table-valued parameters. For this reason, the @IdentityLink table then connects the OrderID values from the @orders parameter to the real OrderID values for the new rows in the PurchaseOrder table. After this step, the @IdentityLink table can facilitate inserting the order details with the actual OrderID that satisfies the foreign key constraint.

This stored procedure can be used from code or from other Transact-SQL calls. See the table-valued parameters section of this paper for a code example. The following Transact-SQL shows how to call the sp\_InsertOrdersBatch.

```

declare @orders as PurchaseOrderTableType
declare @details as PurchaseOrderDetailTableType

INSERT @orders
([OrderID], [OrderDate], [CustomerID], [Status])
VALUES(1, '1/1/2013', 1125, 'Complete'),
(2, '1/13/2013', 348, 'Processing'),
(3, '1/12/2013', 2504, 'Shipped')

INSERT @details
([OrderID], [ProductID], [UnitPrice], [OrderQty])
VALUES(1, 10, $11.50, 1),
(1, 12, $1.58, 1),
(2, 23, $2.57, 2),
(3, 4, $10.00, 1)

exec sp_InsertOrdersBatch @orders, @details

```

This solution allows each batch to use a set of OrderID values that begin at 1. These temporary OrderID values describe the relationships in the batch, but the actual OrderID values are determined at the time of the insert operation. You can run the same statements in the previous example repeatedly and generate unique orders in the database. For this reason, consider adding more code or database logic that prevents duplicate orders when using this batching technique.

This example demonstrates that even more complex database operations, such as master-detail operations, can be batched using table-valued parameters.

## UPSERT

Another batching scenario involves simultaneously updating existing rows and inserting new rows. This operation is sometimes referred to as an “UPSERT” (update + insert) operation. Rather than making separate calls to INSERT and UPDATE, the MERGE statement is best suited to this task. The MERGE statement can perform both insert and update operations in a single call.

Table-valued parameters can be used with the MERGE statement to perform updates and inserts. For example, consider a simplified Employee table that contains the following columns: EmployeeID, FirstName, LastName, SocialSecurityNumber:

```

CREATE TABLE [dbo].[Employee](
[EmployeeID] [int] IDENTITY(1,1) NOT NULL,
[FirstName] [nvarchar](50) NOT NULL,
[LastName] [nvarchar](50) NOT NULL,
[SocialSecurityNumber] [nvarchar](50) NOT NULL,
CONSTRAINT [PrimaryKey_Employee] PRIMARY KEY CLUSTERED
([EmployeeID] ASC)

```

In this example, you can use the fact that the SocialSecurityNumber is unique to perform a MERGE of multiple employees. First, create the user-defined table type:

```

CREATE TYPE EmployeeTableType AS TABLE
(Employee_ID INT,
 FirstName NVARCHAR(50),
 LastName NVARCHAR(50),
 SocialSecurityNumber NVARCHAR(50));
GO

```

Next, create a stored procedure or write code that uses the MERGE statement to perform the update and insert. The following example uses the MERGE statement on a table-valued parameter, @employees, of type EmployeeTableType. The contents of the @employees table are not shown here.

```

MERGE Employee AS target
USING (SELECT [FirstName], [LastName], [SocialSecurityNumber] FROM @employees)
AS source ([FirstName], [LastName], [SocialSecurityNumber])
ON (target.[SocialSecurityNumber] = source.[SocialSecurityNumber])
WHEN MATCHED THEN
UPDATE SET
target.FirstName = source.FirstName,
target.LastName = source.LastName
WHEN NOT MATCHED THEN
 INSERT ([FirstName], [LastName], [SocialSecurityNumber])
 VALUES (source.[FirstName], source.[LastName], source.[SocialSecurityNumber]);

```

For more information, see the documentation and examples for the MERGE statement. Although the same work could be performed in a multiple-step stored procedure call with separate INSERT and UPDATE operations, the MERGE statement is more efficient. Database code can also construct Transact-SQL calls that use the MERGE statement directly without requiring two database calls for INSERT and UPDATE.

## Recommendation summary

The following list provides a summary of the batching recommendations discussed in this article:

- Use buffering and batching to increase the performance and scalability of SQL Database applications.
- Understand the tradeoffs between batching/buffering and resiliency. During a role failure, the risk of losing an unprocessed batch of business-critical data might outweigh the performance benefit of batching.
- Attempt to keep all calls to the database within a single datacenter to reduce latency.
- If you choose a single batching technique, table-valued parameters offer the best performance and flexibility.
- For the fastest insert performance, follow these general guidelines but test your scenario:
  - For < 100 rows, use a single parameterized INSERT command.
  - For < 1000 rows, use table-valued parameters.
  - For >= 1000 rows, use SqlBulkCopy.
- For update and delete operations, use table-valued parameters with stored procedure logic that determines the correct operation on each row in the table parameter.
- Batch size guidelines:
  - Use the largest batch sizes that make sense for your application and business requirements.
  - Balance the performance gain of large batches with the risks of temporary or catastrophic failures.
    - What is the consequence of retries or loss of the data in the batch?
  - Test the largest batch size to verify that SQL Database does not reject it.
  - Create configuration settings that control batching, such as the batch size or the buffering time window.
    - These settings provide flexibility. You can change the batching behavior in production without redeploying the cloud service.
- Avoid parallel execution of batches that operate on a single table in one database. If you do choose to divide a single batch across multiple worker threads, run tests to determine the ideal number of threads. After an unspecified threshold, more threads will decrease performance rather than increase it.
- Consider buffering on size and time as a way of implementing batching for more scenarios.

## Next steps

This article focused on how database design and coding techniques related to batching can improve your application performance and scalability. But this is just one factor in your overall strategy. For more ways to improve performance and scalability, see [Azure SQL Database performance guidance for single databases](#) and [Price and performance considerations for an elastic pool](#).

# Troubleshooting transient connection errors to SQL Database

1/15/2020 • 14 minutes to read • [Edit Online](#)

This article describes how to prevent, troubleshoot, diagnose, and mitigate connection errors and transient errors that your client application encounters when it interacts with Azure SQL Database. Learn how to configure retry logic, build the connection string, and adjust other connection settings.

## Transient errors (transient faults)

A transient error, also known as a transient fault, has an underlying cause that soon resolves itself. An occasional cause of transient errors is when the Azure system quickly shifts hardware resources to better load-balance various workloads. Most of these reconfiguration events finish in less than 60 seconds. During this reconfiguration time span, you might have connectivity issues to SQL Database. Applications that connect to SQL Database should be built to expect these transient errors. To handle them, implement retry logic in their code instead of surfacing them to users as application errors.

If your client program uses ADO.NET, your program is told about the transient error by the throw of **SqlException**.

### Connection vs. command

Retry the SQL connection or establish it again, depending on the following:

- **A transient error occurs during a connection try**

After a delay of several seconds, retry the connection.

- **A transient error occurs during a SQL query command**

Do not immediately retry the command. Instead, after a delay, freshly establish the connection. Then retry the command.

## Retry logic for transient errors

Client programs that occasionally encounter a transient error are more robust when they contain retry logic. When your program communicates with SQL Database through third-party middleware, ask the vendor whether the middleware contains retry logic for transient errors.

### Principles for retry

- If the error is transient, retry to open a connection.
- Do not directly retry a SQL `SELECT` statement that failed with a transient error. Instead, establish a fresh connection, and then retry the `SELECT`.
- When a SQL `UPDATE` statement fails with a transient error, establish a fresh connection before you retry the `UPDATE`. The retry logic must ensure that either the entire database transaction finished or that the entire transaction is rolled back.

### Other considerations for retry

- A batch program that automatically starts after work hours and finishes before morning can afford to be very patient with long time intervals between its retry attempts.
- A user interface program should account for the human tendency to give up after too long a wait. The solution

must not retry every few seconds, because that policy can flood the system with requests.

## Interval increase between retries

We recommend that you wait for 5 seconds before your first retry. Retrying after a delay shorter than 5 seconds risks overwhelming the cloud service. For each subsequent retry, the delay should grow exponentially, up to a maximum of 60 seconds.

For a discussion of the blocking period for clients that use ADO.NET, see [SQL Server connection pooling \(ADO.NET\)](#).

You also might want to set a maximum number of retries before the program self-terminates.

## Code samples with retry logic

Code examples with retry logic are available at:

- [Connect resiliently to SQL with ADO.NET](#)
- [Connect resiliently to SQL with PHP](#)

## Test your retry logic

To test your retry logic, you must simulate or cause an error that can be corrected while your program is still running.

### Test by disconnecting from the network

One way you can test your retry logic is to disconnect your client computer from the network while the program is running. The error is:

- **SqlException.Number** = 11001
- Message: "No such host is known"

As part of the first retry attempt, you can reconnect your client computer to the network and then attempt to connect.

To make this test practical, unplug your computer from the network before you start your program. Then your program recognizes a runtime parameter that causes the program to:

- Temporarily add 11001 to its list of errors to consider as transient.
- Attempt its first connection as usual.
- After the error is caught, remove 11001 from the list.
- Display a message that tells the user to plug the computer into the network.
- Pause further execution by using either the **Console.ReadLine** method or a dialog with an OK button. The user presses the Enter key after the computer is plugged into the network.
- Attempt again to connect, expecting success.

### Test by misspelling the database name when connecting

Your program can purposely misspell the user name before the first connection attempt. The error is:

- **SqlException.Number** = 18456
- Message: "Login failed for user 'WRONG\_UserName'."

As part of the first retry attempt, your program can correct the misspelling and then attempt to connect.

To make this test practical, your program recognizes a runtime parameter that causes the program to:

- Temporarily add 18456 to its list of errors to consider as transient.
- Purposefully add 'WRONG\_' to the user name.
- After the error is caught, remove 18456 from the list.
- Remove 'WRONG\_' from the user name.

- Attempt again to connect, expecting success.

## .NET SqlConnection parameters for connection retry

If your client program connects to SQL Database by using the .NET Framework class

**System.Data.SqlClient.SqlConnection**, use .NET 4.6.1 or later (or .NET Core) so that you can use its connection retry feature. For more information on the feature, see [this webpage](#).

When you build the [connection string](#) for your **SqlConnection** object, coordinate the values among the following parameters:

- **ConnectRetryCount**: Default is 1. Range is 0 through 255.
- **ConnectRetryInterval**: Default is 10 seconds. Range is 1 through 60.
- **Connection Timeout**: Default is 15 seconds. Range is 0 through 2147483647.

Specifically, your chosen values should make the following equality true: Connection Timeout = ConnectRetryCount \* ConnectionRetryInterval

For example, if the count equals 3 and the interval equals 10 seconds, a timeout of only 29 seconds doesn't give the system enough time for its third and final retry to connect:  $29 < 3 * 10$ .

## Connection vs. command

The **ConnectRetryCount** and **ConnectRetryInterval** parameters let your **SqlConnection** object retry the connect operation without telling or bothering your program, such as returning control to your program. The retries can occur in the following situations:

- `mySqlConnection.Open` method call
- `mySqlConnection.ExecuteNonQuery` method call

There is a subtlety. If a transient error occurs while your *query* is being executed, your **SqlConnection** object doesn't retry the connect operation. It certainly doesn't retry your query. However, **SqlConnection** very quickly checks the connection before sending your query for execution. If the quick check detects a connection problem, **SqlConnection** retries the connect operation. If the retry succeeds, your query is sent for execution.

### Should ConnectRetryCount be combined with application retry logic

Suppose your application has robust custom retry logic. It might retry the connect operation four times. If you add **ConnectRetryInterval** and **ConnectRetryCount** = 3 to your connection string, you will increase the retry count to  $4 * 3 = 12$  retries. You might not intend such a high number of retries.

## Connections to SQL Database

### Connection: Connection string

The connection string that's necessary to connect to SQL Database is slightly different from the string used to connect to SQL Server. You can copy the connection string for your database from the [Azure portal](#).

### Obtain the connection string from the Azure portal

Use the [Azure portal](#) to obtain the connection string that's necessary for your client program to interact with Azure SQL Database.

1. Select **All services > SQL databases**.
2. Enter the name of your database into the filter text box near the upper left of the **SQL databases** blade.
3. Select the row for your database.
4. After the blade appears for your database, for visual convenience select the **Minimize** buttons to collapse

the blades you used for browsing and database filtering.

5. On the blade for your database, select **Show database connection strings**.
6. Copy the appropriate connection string. i.e. If you intend to use the ADO.NET connection library, copy the appropriate string from the **ADO.NET** tab.

The screenshot shows the 'Connection strings' blade for a database named 'mySampleDatabase'. The 'ADO.NET' tab is active, displaying three connection string configurations:

- ADO.NET (SQL authentication)**:  
Server=tcp:nntempserver.database.windows.net,1433;Initial Catalog=mySampleDatabase;Persist Security Info=False;User ID=[your\_username];Password=[your\_password];MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
- ADO.NET (Active Directory password authentication)**:  
Server=tcp:nntempserver.database.windows.net,1433;Initial Catalog=mySampleDatabase;Persist Security Info=False;User ID=[your\_username];Password=[your\_password];MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Authentication="Active Directory Password";
- ADO.NET (Active Directory integrated authentication)**:  
Server=tcp:nntempserver.database.windows.net,1433;Initial Catalog=mySampleDatabase;Persist Security Info=False;User ID=[your\_username];MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Authentication="Active Directory Integrated";

A 'Download ADO.NET driver for SQL server' link is also present at the bottom of the blade.

7. Edit the connection string as needed. i.e. Insert your password into the connection string, or remove "@<servername>" from the username if the username or server name are too long.
8. In one format or another, paste the connection string information into your client program code.

For more information, see [Connection strings and configuration files](#).

### Connection: IP address

You must configure the SQL Database server to accept communication from the IP address of the computer that hosts your client program. To set up this configuration, edit the firewall settings through the [Azure portal](#).

If you forget to configure the IP address, your program fails with a handy error message that states the necessary IP address.

1. Sign in to the [Azure portal](#).
2. In the list on the left, select **All services**.
3. Scroll and select **SQL servers**.

The screenshot shows the 'All services' blade in the Microsoft Azure portal. The 'SQL servers' service is selected, indicated by a blue border around its icon and name. Other services listed include:

- Azure Cosmos DB
- SQL databases
- Azure Database for PostgreSQL servers
- SQL data warehouses
- Redis Caches
- Data factories
- Virtual clusters
- Elastic Job agents

A search bar at the top is labeled 'Search resources, services, and docs'. A filter bar below it has a 'Filter' input field and a 'By category' dropdown. Navigation icons for expanding and collapsing categories are also visible.

4. In the filter text box, start typing the name of your server. Your row is displayed.

5. Select the row for your server. A blade for your server is displayed.

6. On your server blade, select **Settings**.

7. Select **Firewall**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a tree view with 'demo-pool-server' selected. Under 'demo-pool-server', 'Firewalls and virtual networks' is highlighted with a red box. The main content area shows a table for defining IP ranges. The first row of the table is also highlighted with a red box. The table columns are 'RULE NAME', 'START IP', and 'END IP'. Below the table, there's a note about VNET/Subnet rules.

8. Select **Add Client IP**. Type a name for your new rule in the first text box.

9. Type in the low and high IP address values for the range you want to enable.

- It can be handy to have the low value end with **.0** and the high value end with **.255**.

10. Select **Save**.

For more information, see [Configure firewall settings on SQL Database](#).

### Connection: Ports

Typically, you need to ensure that only port 1433 is open for outbound communication on the computer that hosts your client program.

For example, when your client program is hosted on a Windows computer, you can use Windows Firewall on the host to open port 1433.

1. Open Control Panel.

2. Select **All Control Panel Items > Windows Firewall > Advanced Settings > Outbound Rules > Actions > New Rule**.

If your client program is hosted on an Azure virtual machine (VM), read [Ports beyond 1433 for ADO.NET 4.5 and SQL Database](#).

For background information about configuration of ports and IP addresses, see [Azure SQL Database firewall](#).

### Connection: ADO.NET 4.6.2 or later

If your program uses ADO.NET classes like **System.Data.SqlClient.SqlConnection** to connect to SQL Database, we recommend that you use .NET Framework version 4.6.2 or later.

#### Starting with ADO.NET 4.6.2

- The connection open attempt to be retried immediately for Azure SQL databases, thereby improving the performance of cloud-enabled apps.

#### Starting with ADO.NET 4.6.1

- For SQL Database, reliability is improved when you open a connection by using the **SqlConnection.Open** method. The **Open** method now incorporates best-effort retry mechanisms in response to transient faults for certain errors within the connection timeout period.
- Connection pooling is supported, which includes an efficient verification that the connection object it gives your program is functioning.

When you use a connection object from a connection pool, we recommend that your program temporarily closes the connection when it's not immediately in use. It's not expensive to reopen a connection, but it is to create a new connection.

If you use ADO.NET 4.0 or earlier, we recommend that you upgrade to the latest ADO.NET. As of August 2018, you can [download ADO.NET 4.6.2](#).

## Diagnostics

### Diagnostics: Test whether utilities can connect

If your program fails to connect to SQL Database, one diagnostic option is to try to connect with a utility program. Ideally, the utility connects by using the same library that your program uses.

On any Windows computer, you can try these utilities:

- SQL Server Management Studio (ssms.exe), which connects by using ADO.NET
- `sqlcmd.exe`, which connects by using [ODBC](#)

After your program is connected, test whether a short SQL SELECT query works.

### Diagnostics: Check the open ports

If you suspect that connection attempts fail due to port issues, you can run a utility on your computer that reports on the port configurations.

On Linux, the following utilities might be helpful:

- `netstat -nap`
- `nmap -sS -o 127.0.0.1` : Change the example value to be your IP address.

On Windows, the [PortQry.exe](#) utility might be helpful. Here's an example execution that queried the port situation on a SQL Database server and that was run on a laptop computer:

```
[C:\Users\johndoe\]
>> portqry.exe -n johndoesvr9.database.windows.net -p tcp -e 1433

Querying target system called: johndoesvr9.database.windows.net

Attempting to resolve name to IP address...
Name resolved to 23.100.117.95

querying...
TCP port 1433 (ms-sql-s service): LISTENING

[C:\Users\johndoe\]
>>
```

### Diagnostics: Log your errors

An intermittent problem is sometimes best diagnosed by detection of a general pattern over days or weeks.

Your client can assist in a diagnosis by logging all errors it encounters. You might be able to correlate the log entries with error data that SQL Database logs itself internally.

Enterprise Library 6 (EntLib60) offers .NET managed classes to assist with logging. For more information, see [5 - As easy as falling off a log: Use the Logging Application Block](#).

## Diagnostics: Examine system logs for errors

Here are some Transact-SQL SELECT statements that query error logs and other information.

QUERY OF LOG	DESCRIPTION
<pre>SELECT e.* FROM sys.event_log AS e WHERE e.database_name = 'myDbName' AND e.event_category = 'connectivity' AND 2 &gt;= DateDiff     (hour, e.end_time, GetUtcDate()) ORDER BY e.event_category,         e.event_type, e.end_time;</pre>	<p>The <a href="#">sys.event_log</a> view offers information about individual events, which includes some that can cause transient errors or connectivity failures.</p> <p>Ideally, you can correlate the <b>start_time</b> or <b>end_time</b> values with information about when your client program experienced problems.</p> <p>You must connect to the <i>master</i> database to run this query.</p>
<pre>SELECT c.* FROM sys.database_connection_stats AS c WHERE c.database_name = 'myDbName' AND 24 &gt;= DateDiff     (hour, c.end_time, GetUtcDate()) ORDER BY c.end_time;</pre>	<p>The <a href="#">sys.database_connection_stats</a> view offers aggregated counts of event types for additional diagnostics.</p> <p>You must connect to the <i>master</i> database to run this query.</p>

## Diagnostics: Search for problem events in the SQL Database log

You can search for entries about problem events in the SQL Database log. Try the following Transact-SQL SELECT statement in the *master* database:

```
SELECT
 object_name
 ,CAST(f.event_data as XML).value
 ('(/event/@timestamp)[1]', 'datetime2') AS [timestamp]
 ,CAST(f.event_data as XML).value
 ('(/event/data[@name="error"]/value)[1]', 'int') AS [error]
 ,CAST(f.event_data as XML).value
 ('(/event/data[@name="state"]/value)[1]', 'int') AS [state]
 ,CAST(f.event_data as XML).value
 ('(/event/data[@name="is_success"]/value)[1]', 'bit') AS [is_success]
 ,CAST(f.event_data as XML).value
 ('(/event/data[@name="database_name"]/value)[1]', 'sysname') AS [database_name]
FROM
 sys.fn_xe_telemetry_blob_target_read_file('el', null, null, null) AS f
WHERE
 object_name != 'login_event' -- Login events are numerous.
 and
 '2015-06-21' < CAST(f.event_data as XML).value
 ('(/event/@timestamp)[1]', 'datetime2')
ORDER BY
 [timestamp] DESC
;
```

### A few returned rows from `sys.fn_xe_telemetry_blob_target_read_file`

The following example shows what a returned row might look like. The null values shown are often not null in other rows.

object_name	timestamp	error	state	is_success	database_name
database_xml_deadlock_report	2015-10-16 20:28:01.0090000	NULL	NULL	NULL	AdventureWorks

# Enterprise Library 6

Enterprise Library 6 (EntLib60) is a framework of .NET classes that helps you implement robust clients of cloud services, one of which is the SQL Database service. To locate topics dedicated to each area in which EntLib60 can assist, see [Enterprise Library 6 - April 2013](#).

Retry logic for handling transient errors is one area in which EntLib60 can assist. For more information, see [4 - Perseverance, secret of all triumphs: Use the Transient Fault Handling Application Block](#).

## NOTE

The source code for EntLib60 is available for public download from the [Download Center](#). Microsoft has no plans to make further feature updates or maintenance updates to EntLib.

## EntLib60 classes for transient errors and retry

The following EntLib60 classes are particularly useful for retry logic. All these classes are found in or under the namespace **Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling**.

In the namespace **Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling**:

- **RetryPolicy** class
  - **ExecuteAction** method
- **ExponentialBackoff** class
- **SqlDatabaseTransientErrorDetectionStrategy** class
- **ReliableSqlConnection** class
  - **ExecuteCommand** method

In the namespace **Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling.TestTools**:

- **AlwaysTransientErrorDetectionStrategy** class
- **NeverTransientErrorDetectionStrategy** class

Here are some links to information about EntLib60:

- Free book download: [Developer's Guide to Microsoft Enterprise Library, 2nd edition](#).
- Best practices: [Retry general guidance](#) has an excellent in-depth discussion of retry logic.
- NuGet download: [Enterprise Library - Transient Fault Handling Application Block 6.0](#).

## EntLib60: The logging block

- The logging block is a highly flexible and configurable solution that you can use to:
  - Create and store log messages in a wide variety of locations.
  - Categorize and filter messages.
  - Collect contextual information that is useful for debugging and tracing, as well as for auditing and general logging requirements.
- The logging block abstracts the logging functionality from the log destination so that the application code is consistent, irrespective of the location and type of the target logging store.

For more information, see [5 - As easy as falling off a log: Use the Logging Application Block](#).

## EntLib60 IsTransient method source code

Next, from the **SqlDatabaseTransientErrorDetectionStrategy** class, is the C# source code for the **IsTransient** method. The source code clarifies which errors were considered transient and worthy of retry, as of April 2013.

```

public bool IsTransient(Exception ex)
{
 if (ex != null)
 {
 SqlException sqlException;
 if ((sqlException = ex as SqlException) != null)
 {
 // Enumerate through all errors found in the exception.
 foreach (SqlError err in sqlException.Errors)
 {
 switch (err.Number)
 {
 // SQL Error Code: 40501
 // The service is currently busy. Retry the request after 10 seconds.
 // Code: (reason code to be decoded).
 case ThrottlingCondition.ThrottlingErrorNumber:
 // Decode the reason code from the error message to
 // determine the grounds for throttling.
 var condition = ThrottlingCondition.FromError(err);

 // Attach the decoded values as additional attributes to
 // the original SQL exception.
 sqlException.Data[condition.ThrottlingMode.GetType().Name] =
 condition.ThrottlingMode.ToString();
 sqlException.Data[condition.GetType().Name] = condition;

 return true;

 case 10928:
 case 10929:
 case 10053:
 case 10054:
 case 10060:
 case 40197:
 case 40540:
 case 40613:
 case 40143:
 case 233:
 case 64:
 // DBNETLIB Error Code: 20
 // The instance of SQL Server you attempted to connect to
 // does not support encryption.
 case (int)ProcessNetLibErrorCode.EncryptionNotSupported:
 return true;
 }
 }
 }
 else if (ex is TimeoutException)
 {
 return true;
 }
 else
 {
 EntityException entityException;
 if ((entityException = ex as EntityException) != null)
 {
 return this.IsTransient(entityException.InnerException);
 }
 }
 }

 return false;
}

```

## Next steps

- Connection libraries for SQL Database and SQL Server
- SQL Server connection pooling (ADO.NET)
- *Retrying* is an Apache 2.0 licensed general-purpose retrying library, written in Python, to simplify the task of adding retry behavior to just about anything.

# DNS alias for Azure SQL Database

11/7/2019 • 5 minutes to read • [Edit Online](#)

Azure SQL Database has a Domain Name System (DNS) server. PowerShell and REST APIs accept [calls to create and manage DNS aliases](#) for your SQL Database server name.

A *DNS alias* can be used in place of the Azure SQL Database server name. Client programs can use the alias in their connection strings. The DNS alias provides a translation layer that can redirect your client programs to different servers. This layer spares you the difficulties of having to find and edit all the clients and their connection strings.

Common uses for a DNS alias include the following cases:

- Create an easy to remember name for an Azure SQL Server.
- During initial development, your alias can refer to a test SQL Database server. When the application goes live, you can modify the alias to refer to the production server. The transition from test to production does not require any modification to the configurations several clients that connect to the database server.
- Suppose the only database in your application is moved to another SQL Database server. Here you can modify the alias without having to modify the configurations of several clients.
- During a regional outage you use geo-restore to recover your database in a different server and region. You can modify your existing alias to point to the new server so that the existing client application could re-connect to it.

## Domain Name System (DNS) of the Internet

The Internet relies on the DNS. The DNS translates your friendly names into the name of your Azure SQL Database server.

## Scenarios with one DNS alias

Suppose you need to switch your system to a new Azure SQL Database server. In the past you needed to find and update every connection string in every client program. But now, if the connection strings use a DNS alias, only an alias property must be updated.

The DNS alias feature of Azure SQL Database can help in the following scenarios:

### Test to production

When you start developing the client programs, have them use a DNS alias in their connection strings. You make the properties of the alias point to a test version of your Azure SQL Database server.

Later when the new system goes live in production, you can update the properties of the alias to point to the production SQL Database server. No change to the client programs is necessary.

### Cross-region support

A disaster recovery might shift your SQL Database server to a different geographic region. For a system that was using a DNS alias, the need to find and update all the connection strings for all clients can be avoided. Instead, you can update an alias to refer to the new SQL Database server that now hosts your database.

## Properties of a DNS alias

The following properties apply to each DNS alias for your SQL Database server:

- *Unique name*: Each alias name you create is unique across all Azure SQL Database servers, just as server names are.
- *Server is required*: A DNS alias cannot be created unless it references exactly one server, and the server must already exist. An updated alias must always reference exactly one existing server.
  - When you drop a SQL Database server, the Azure system also drops all DNS aliases that refer to the server.
- *Not bound to any region*: DNS aliases are not bound to a region. Any DNS aliases can be updated to refer to an Azure SQL Database server that resides in any geographic region.
  - However, when updating an alias to refer to another server, both servers must exist in the same Azure subscription.
- *Permissions*: To manage a DNS alias, the user must have *Server Contributor* permissions, or higher. For more information, see [Get started with Role-Based Access Control in the Azure portal](#).

## Manage your DNS aliases

Both REST APIs and PowerShell cmdlets are available to enable you to programmatically manage your DNS aliases.

### REST APIs for managing your DNS aliases

The documentation for the REST APIs is available near the following web location:

- [Azure SQL Database REST API](#)

Also, the REST APIs can be seen in GitHub at:

- [Azure SQL Database server, DNS alias REST APIs](#)

### PowerShell for managing your DNS aliases

#### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

#### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

PowerShell cmdlets are available that call the REST APIs.

A code example of PowerShell cmdlets being used to manage DNS aliases is documented at:

- [PowerShell for DNS Alias to Azure SQL Database](#)

The cmdlets used in the code example are the following:

- [New-AzSqlServerDNSAlias](#): Creates a new DNS alias in the Azure SQL Database service system. The alias refers to Azure SQL Database server 1.
- [Get-AzSqlServerDNSAlias](#): Get and list all the DNS aliases that are assigned to SQL DB server 1.
- [Set-AzSqlServerDNSAlias](#): Modifies the server name that the alias is configured to refer to, from server 1 to SQL DB server 2.

- [Remove-AzSqlServerDNSAlias](#): Remove the DNS alias from SQL DB server 2, by using the name of the alias.

## Limitations during preview

Presently, a DNS alias has the following limitations:

- *Delay of up to 2 minutes*: It takes up to 2 minutes for a DNS alias to be updated or removed.
  - Regardless of any brief delay, the alias immediately stops referring client connections to the legacy server.
- *DNS lookup*: For now, the only authoritative way to check what server a given DNS alias refers to is by performing a [DNS lookup](#).
- *Table auditing is not supported*: You cannot use a DNS alias on an Azure SQL Database server that has *table auditing* enabled on a database.
  - Table auditing is deprecated.
  - We recommend that you move to [Blob Auditing](#).

## Related resources

- [Overview of business continuity with Azure SQL Database](#), including disaster recovery.

## Next steps

- [PowerShell for DNS Alias to Azure SQL Database](#)

# PowerShell for DNS Alias to Azure SQL Database

11/22/2019 • 3 minutes to read • [Edit Online](#)

This article provides a PowerShell script that demonstrates how you can manage a DNS alias for Azure SQL Database.

## NOTE

This article has been updated to use either the Azure PowerShell Az module or Azure CLI. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020.

To learn more about the Az module and AzureRM compatibility, see [Introducing the Azure PowerShell Az module](#). For installation instructions, see [Install Azure PowerShell](#) or [Install Azure CLI](#).

## DNS alias in connection string

To connect a particular Azure SQL Database server, a client such as SQL Server Management Studio (SSMS) can provide the DNS alias name instead of the true server name. In the following example server string, the alias *any-unique-alias-name* replaces the first dot-delimited node in the four node server string:

```
<yourServer>.database.windows.net
```

## Prerequisites

If you want to run the demo PowerShell script given in this article, the following prerequisites apply:

- An Azure subscription and account, for free trial, see [Azure trials](#)
- Two Azure SQL database servers

## Example

The following code example starts by assigning literal values to several variables.

To run the code, edit the placeholder values to match real values in your system.

- [PowerShell](#)
- [Azure CLI](#)

The cmdlets used are the following:

- [New-AzSqlServerDNSAlias](#): Creates a DNS alias in the Azure SQL Database service system. The alias refers to database server 1.
- [Get-AzSqlServerDNSAlias](#): Get and list all the aliases assigned to SQL DB server 1.
- [Set-AzSqlServerDNSAlias](#): Modifies the server name that the alias is configured to refer to, from server 1 to server 2.
- [Remove-AzSqlServerDNSAlias](#): Remove the alias from database server 2, by using the name of the alias.

To install or upgrade, see [Install Azure PowerShell module](#).

Use `Get-Module -ListAvailable Az` in `powershell_ise.exe`, to find the version.

```
$subscriptionName = '<subscriptionName>';
$sqlServerDnsAliasName = '<aliasName>';
$resourceGroupName = '<resourceGroupName>';
$sqlServerName = '<sqlServerName>';
$resourceGroupName2 = '<resourceGroupNameTwo>'; # can be same or different than $resourceGroupName
$sqlServerName2 = '<sqlServerNameTwo>'; # must be different from $sqlServerName.

login to Azure
Connect-AzAccount -SubscriptionName $subscriptionName;
$subscriptionId = Get-AzSubscription -SubscriptionName $subscriptionName;

Write-Host 'Assign an alias to server 1...';
New-AzSqlServerDnsAlias -ResourceGroupName $resourceGroupName -ServerName $sqlServerName `
 -Name $sqlServerDnsAliasName;

Write-Host 'Get the aliases assigned to server 1...';
Get-AzSqlServerDnsAlias -ResourceGroupName $resourceGroupName -ServerName $sqlServerName;

Write-Host 'Move the alias from server 1 to server 2...';
Set-AzSqlServerDnsAlias -ResourceGroupName $resourceGroupName2 -TargetServerName $sqlServerName2 `
 -Name $sqlServerDnsAliasName `
 -SourceServerResourceGroup $resourceGroupName -SourceServerName $sqlServerName `
 -SourceServerSubscriptionId $subscriptionId.Id;

Write-Host 'Get the aliases assigned to server 2...';
Get-AzSqlServerDnsAlias -ResourceGroupName $resourceGroupName2 -ServerName $sqlServerName2;

Write-Host 'Remove the alias from server 2...';
Remove-AzSqlServerDnsAlias -ResourceGroupName $resourceGroupName2 -ServerName $sqlServerName2 `
 -Name $sqlServerDnsAliasName;
```

## Next steps

For a full explanation of the DNS alias feature for SQL Database, see [DNS alias for Azure SQL database](#).

# Ports beyond 1433 for ADO.NET 4.5

11/7/2019 • 2 minutes to read • [Edit Online](#)

This topic describes the Azure SQL Database connection behavior for clients that use ADO.NET 4.5 or a later version.

## IMPORTANT

For information about connectivity architecture, see [Azure SQL Database connectivity architecture](#).

## Outside vs inside

For connections to Azure SQL Database, we must first ask whether your client program runs *outside* or *inside* the Azure cloud boundary. The subsections discuss two common scenarios.

### **Outside: Client runs on your desktop computer**

Port 1433 is the only port that must be open on your desktop computer that hosts your SQL Database client application.

### **Inside: Client runs on Azure**

When your client runs inside the Azure cloud boundary, it uses what we can call a *direct route* to interact with the SQL Database server. After a connection is established, further interactions between the client and database involve no Azure SQL Database Gateway.

The sequence is as follows:

1. ADO.NET 4.5 (or later) initiates a brief interaction with the Azure cloud, and receives a dynamically identified port number.
  - The dynamically identified port number is in the range of 11000-11999.
2. ADO.NET then connects to the SQL Database server directly, with no middleware in between.
3. Queries are sent directly to the database, and results are returned directly to the client.

Ensure that the port ranges of 11000-11999 on your Azure client machine are left available for ADO.NET 4.5 client interactions with SQL Database.

- In particular, ports in the range must be free of any other outbound blockers.
- On your Azure VM, the **Windows Firewall with Advanced Security** controls the port settings.
  - You can use the [firewall's user interface](#) to add a rule for which you specify the **TCP** protocol along with a port range with the syntax like **11000-11999**.

## Version clarifications

This section clarifies the monikers that refer to product versions. It also lists some pairings of versions between products.

### **ADO.NET**

- ADO.NET 4.0 supports the TDS 7.3 protocol, but not 7.4.
- ADO.NET 4.5 and later supports the TDS 7.4 protocol.

## **ODBC**

- Microsoft SQL Server ODBC 11 or above

## **JDBC**

- Microsoft SQL Server JDBC 4.2 or above (JDBC 4.0 actually supports TDS 7.4 but does not implement "redirection")

## Related links

- ADO.NET 4.6 was released on July 20, 2015. A blog announcement from the .NET team is available [here](#).
- ADO.NET 4.5 was released on August 15, 2012. A blog announcement from the .NET team is available [here](#).
  - A blog post about ADO.NET 4.5.1 is available [here](#).
- Microsoft® ODBC Driver 17 for SQL Server® - Windows, Linux, & macOS  
<https://www.microsoft.com/download/details.aspx?id=56567>
- Connect to Azure SQL Database V12 via Redirection  
<https://techcommunity.microsoft.com/t5/DataCAT/Connect-to-Azure-SQL-Database-V12-via-Redirection/ba-p/305362>
- [TDS protocol version list](#)
- [SQL Database Development Overview](#)
- [Azure SQL Database firewall](#)
- [How to: Configure firewall settings on SQL Database](#)

# Connect to SQL Database using C and C++

11/6/2019 • 5 minutes to read • [Edit Online](#)

This post is aimed at C and C++ developers trying to connect to Azure SQL DB. It is broken down into sections so you can jump to the section that best captures your interest.

## Prerequisites for the C/C++ tutorial

Make sure you have the following items:

- An active Azure account. If you don't have one, you can sign up for a [Free Azure Trial](#).
- [Visual Studio](#). You must install the C++ language components to build and run this sample.
- [Visual Studio Linux Development](#). If you are developing on Linux, you must also install the Visual Studio Linux extension.

## Azure SQL Database and SQL Server on virtual machines

Azure SQL is built on Microsoft SQL Server and is designed to provide a high-availability, performant, and scalable service. There are many benefits to using SQL Azure over your proprietary database running on premises. With SQL Azure you don't have to install, set up, maintain, or manage your database but only the content and the structure of your database. Typical things that we worry about with databases like fault tolerance and redundancy are all built in.

Azure currently has two options for hosting SQL server workloads: Azure SQL database, database as a service and SQL server on Virtual Machines (VM). We will not get into detail about the differences between these two except that Azure SQL database is your best bet for new cloud-based applications to take advantage of the cost savings and performance optimization that cloud services provide. If you are considering migrating or extending your on-premises applications to the cloud, SQL server on Azure virtual machine might work out better for you. To keep things simple for this article, let's create an Azure SQL database.

## Data access technologies: ODBC and OLE DB

Connecting to Azure SQL DB is no different and currently there are two ways to connect to databases: ODBC (Open Database connectivity) and OLE DB (Object Linking and Embedding database). In recent years, Microsoft has aligned with [ODBC for native relational data access](#). ODBC is relatively simple, and also much faster than OLE DB. The only caveat here is that ODBC does use an old C-style API.

## Step 1: Creating your Azure SQL Database

See the [getting started page](#) to learn how to create a sample database. Alternatively, you can follow this [short two-minute video](#) to create an Azure SQL database using the Azure portal.

## Step 2: Get connection string

After your Azure SQL database has been provisioned, you need to carry out the following steps to determine connection information and add your client IP for firewall access.

In [Azure portal](#), go to your Azure SQL database ODBC connection string by using the **Show database connection strings** listed as a part of the overview section for your database:

## Essentials ^

Resource group	Server name
Status	Server version
Online	V12
Location	Connection strings
Central US	<a href="#">Show database connection strings</a> 
Subscription name	Pricing tier
	S2 Standard (50 DTUs)

### Database connection strings

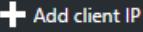
ADO.NET(SQL authentication)  
`MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;` 

ODBC (Includes Node.js) [SQL authentication]  
`Driver={ODBC Driver 13 for SQL Server};Server=tcp:cppsqlldb.database.windows.net,1433;Dat` 

Copy the contents of the **ODBC (Includes Node.js) [SQL authentication]** string. We use this string later to connect from our C++ ODBC command-line interpreter. This string provides details such as the driver, server, and other database connection parameters.

## Step 3: Add your IP to the firewall

Go to the firewall section for your Database server and add your [client IP to the firewall using these steps](#) to make sure we can establish a successful connection:

 Save  Discard  Add client IP

 Connections from the IPs specified below provides access to all the databases in cppsqlldb.

Allow access to Azure services  ON  OFF

Client IP address 167.2. [REDACTED]

At this point, you have configured your Azure SQL DB and are ready to connect from your C++ code.

## Step 4: Connecting from a Windows C/C++ application

You can easily connect to your [Azure SQL DB using ODBC on Windows using this sample](#) that builds with Visual Studio. The sample implements an ODBC command-line interpreter that can be used to connect to our Azure SQL DB. This sample takes either a Database source name file (DSN) file as a command-line argument or the verbose connection string that we copied earlier from the Azure portal. Bring up the property page for this project and paste the connection string as a command argument as shown here:

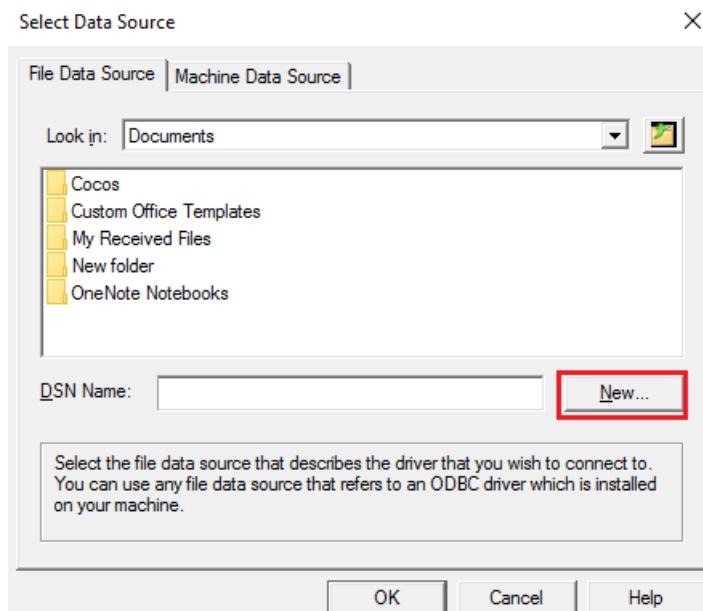
## odbcsql Property Pages

Make sure you provide the right authentication details for your database as a part of that database connection string.

Launch the application to build it. You should see the following window validating a successful connection. You can even run some basic SQL commands like **create table** to validate your database connectivity:

```
C:\ODBC database sample\C+\Debug\odbcsql.exe
[01000] [Microsoft][SQL Server Native Client 11.0][SQL Server]Changed database context to 'democpp'. (5701)
[01000] [Microsoft][SQL Server Native Client 11.0][SQL Server]Changed language setting to us_english. (5703)
[01S00] [Microsoft][SQL Server Native Client 11.0]Invalid connection string attribute (0)
Connected!
Enter SQL commands, type (control)Z to exit
SQL COMMAND>CREATE TABLE Employee(EmployeeID int, LastName varchar(255), FirstName varchar(255));
SQL COMMAND>INSERT INTO Employee(EmployeeID, LastName, FirstName) VALUES(1, 'Asthana', 'Ankit');
1 row affected
SQL COMMAND>SELECT * FROM EMPLOYEE;
| EmployeeID | LastName | FirstName |
| 1 | Asthana | Ankit |
SQL COMMAND>
```

Alternatively, you could create a DSN file using the wizard that is launched when no command arguments are provided. We recommend that you try this option as well. You can use this DSN file for automation and protecting your authentication settings:



Congratulations! You have now successfully connected to Azure SQL using C++ and ODBC on Windows. You can continue reading to do the same for Linux platform as well.

## Step 5: Connecting from a Linux C/C++ application

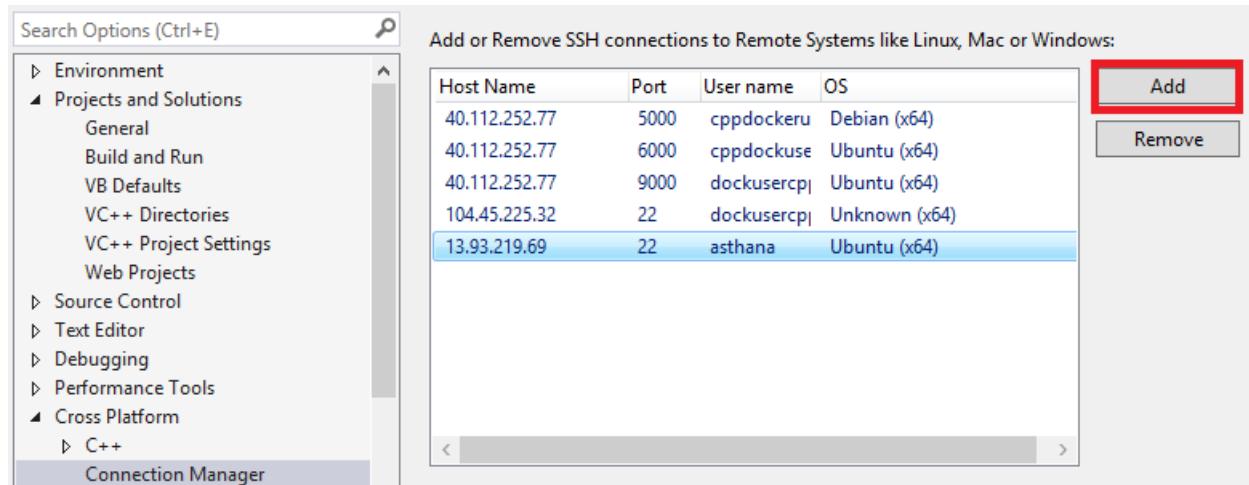
In case you haven't heard the news yet, Visual Studio now allows you to develop C++ Linux application as well. You can read about this new scenario in the [Visual C++ for Linux Development](#) blog. To build for Linux, you need a remote machine where your Linux distro is running. If you don't have one available, you can set one up quickly using [Linux Azure Virtual machines](#).

For this tutorial, let us assume that you have an Ubuntu 16.04 Linux distribution set up. The steps here should also apply to Ubuntu 15.10, Red Hat 6, and Red Hat 7.

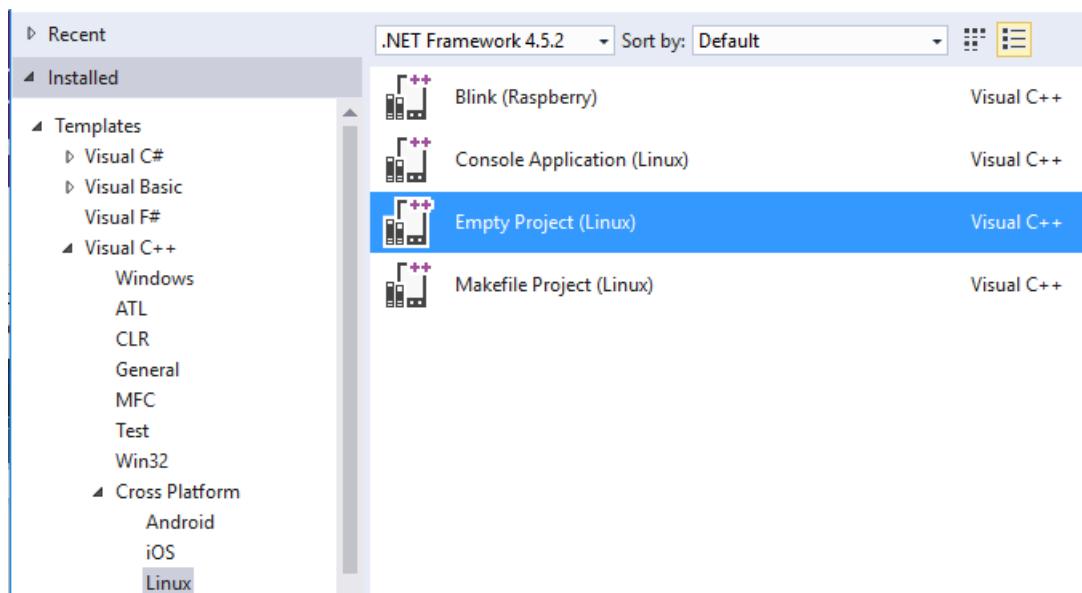
The following steps install the libraries needed for SQL and ODBC for your distro:

```
sudo su
sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/mssql-ubuntu-test/ xenial main" >
/etc/apt/sources.list.d/mssqlpreview.list'
sudo apt-key adv --keyserver apt-mo.trafficmanager.net --recv-keys 417A0893
apt-get update
apt-get install msodbcsql
apt-get install unixodbc-dev-utf16 #this step is optional but recommended*
```

Launch Visual Studio. Under Tools -> Options -> Cross Platform -> Connection Manager, add a connection to your Linux box:



After connection over SSH is established, create an Empty project (Linux) template:

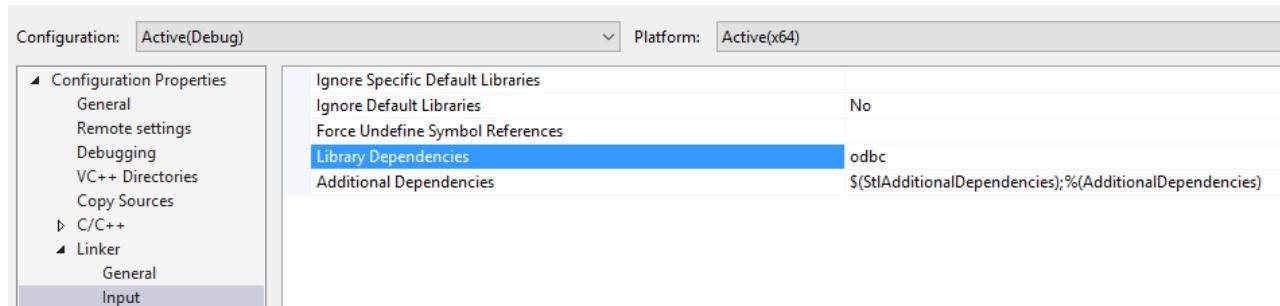


You can then add a [new C source file](#) and replace it with this content. Using the ODBC APIs `SQLAllocHandle`, `SQLSetConnectAttr`, and `SQLDriverConnect`, you should be able to initialize and establish a connection to your database. Like with the Windows ODBC sample, you need to replace the `SQLDriverConnect` call with the details

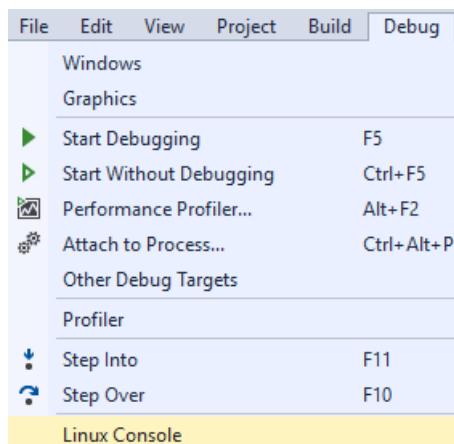
from your database connection string parameters copied from the Azure portal previously.

```
retcode = SQLDriverConnect(
 hdbc, NULL, "Driver=ODBC Driver 13 for SQL"
 "Server;Server=<yourserver>;Uid=<yourusername>;Pwd=<"
 "<yourpassword>;database=<yourdatabase>",
 SQL_NTS, outstr, sizeof(outstr), &outstrlen, SQL_DRIVER_NOPROMPT);
```

The last thing to do before compiling is to add **odbc** as a library dependency:



To launch your application, bring up the Linux Console from the **Debug** menu:



If your connection was successful, you should now see the current database name printed in the Linux Console:

```
Linux Console Window
Process /home/asthana/projects/cplinuxsample/bin/x64/Debug/cplinuxsample.out created; pid = 10037
Listening on port 4444
Remote debugging from host 127.0.0.1
Database name connected to: democpp
Complete.

Child exited with status 0
```

Congratulations! You have successfully completed the tutorial and can now connect to your Azure SQL DB from C++ on Windows and Linux platforms.

## Get the complete C/C++ tutorial solution

You can find the GetStarted solution that contains all the samples in this article at GitHub:

- [ODBC C++ Windows sample](#), Download the Windows C++ ODBC Sample to connect to Azure SQL
- [ODBC C++ Linux sample](#), Download the Linux C++ ODBC Sample to connect to Azure SQL

## Next steps

- Review the [SQL Database Development Overview](#)
- More information on the [ODBC API Reference](#)

## Additional resources

- [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#)
- Explore all the [capabilities of SQL Database](#)

# Connect Excel to a single database in Azure SQL database and create a report

11/7/2019 • 4 minutes to read • [Edit Online](#)

Connect Excel to a single database in Azure SQL Database and import data and create tables and charts based on values in the database. In this tutorial you will set up the connection between Excel and a database table, save the file that stores data and the connection information for Excel, and then create a pivot chart from the database values.

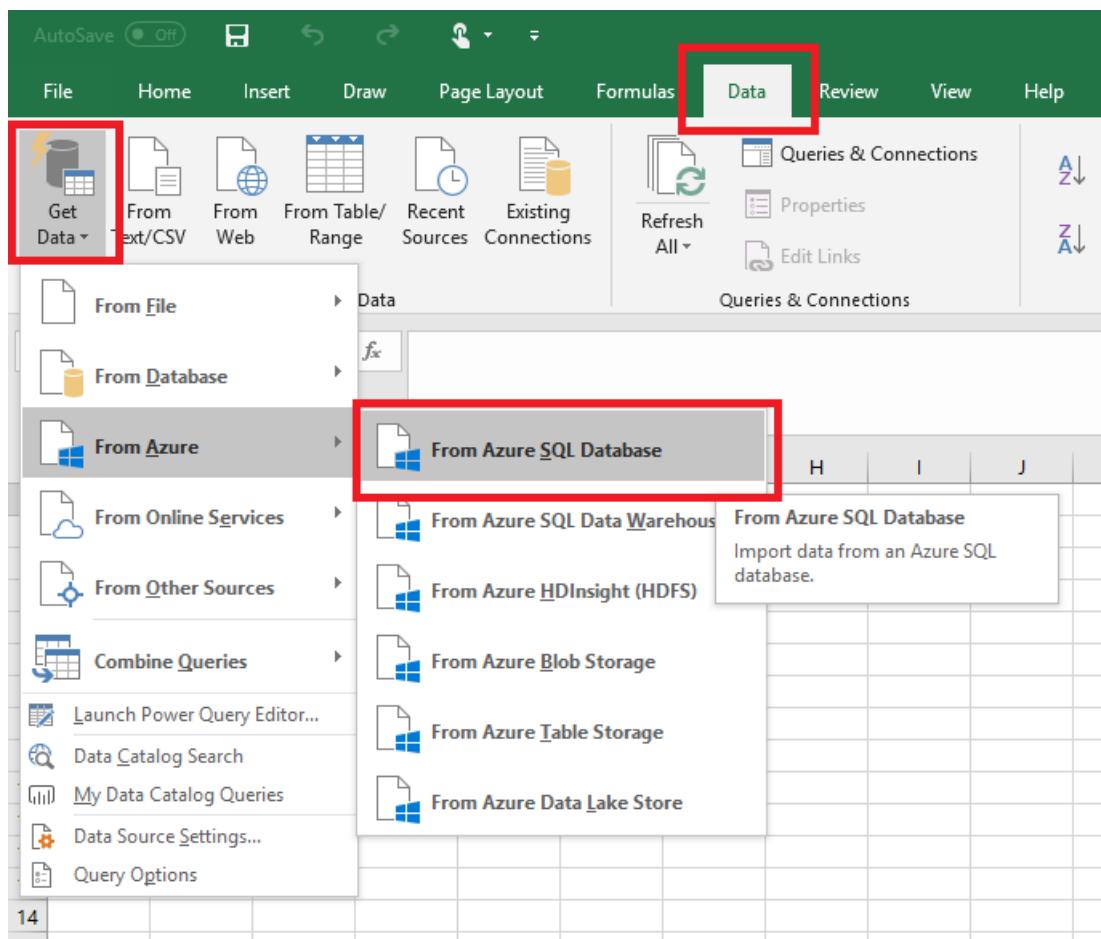
You'll need a single database before you get started. If you don't have one, see [Create a single database](#) and [Create server-level IP firewall](#) to get a single database with sample data up and running in a few minutes.

In this article, you'll import sample data into Excel from that article, but you can follow similar steps with your own data.

You'll also need a copy of Excel. This article uses [Microsoft Excel 2016](#).

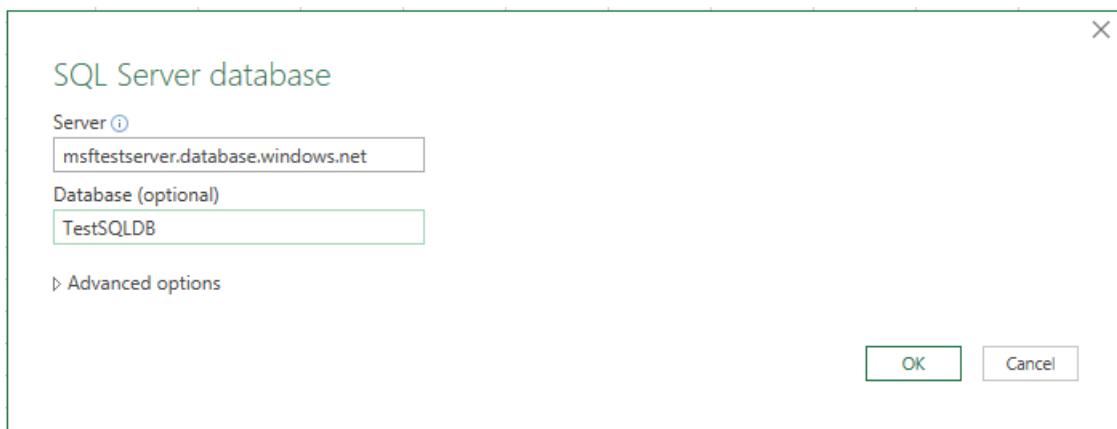
## Connect Excel to a SQL database and load data

1. To connect Excel to SQL database, open Excel and then create a new workbook or open an existing Excel workbook.
2. In the menu bar at the top of the page, select the **Data** tab, select **Get Data**, select From Azure, and then select **From Azure SQL Database**.

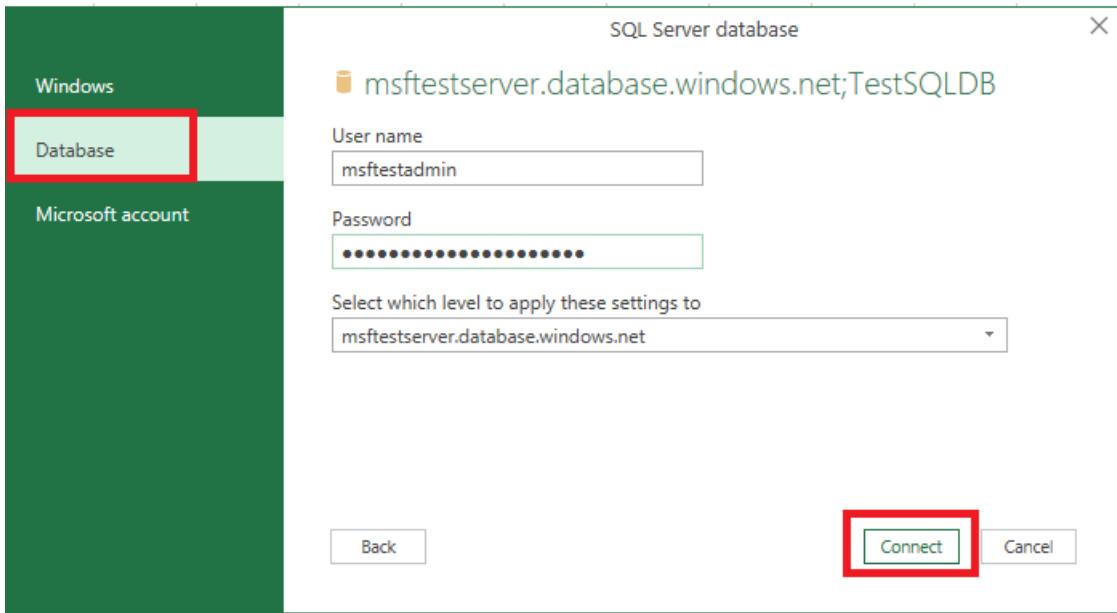


The Data Connection Wizard opens.

3. In the **Connect to Database Server** dialog box, type the SQL Database **Server name** you want to connect to in the form <servername>.database.windows.net. For example, **msftestserver.database.windows.net**. Optionally, enter in the name of your database. Select **OK** to open the credentials window.



4. In the **SQL Server Database** dialog box, select **Database** on the left side, and then enter in your **User Name** and **Password** for the SQL Database server you want to connect to. Select **Connect** to open the **Navigator**.



#### TIP

Depending on your network environment, you may not be able to connect or you may lose the connection if the SQL Database server doesn't allow traffic from your client IP address. Go to the [Azure portal](#), click SQL servers, click your server, click firewall under settings and add your client IP address. See [How to configure firewall settings](#) for details.

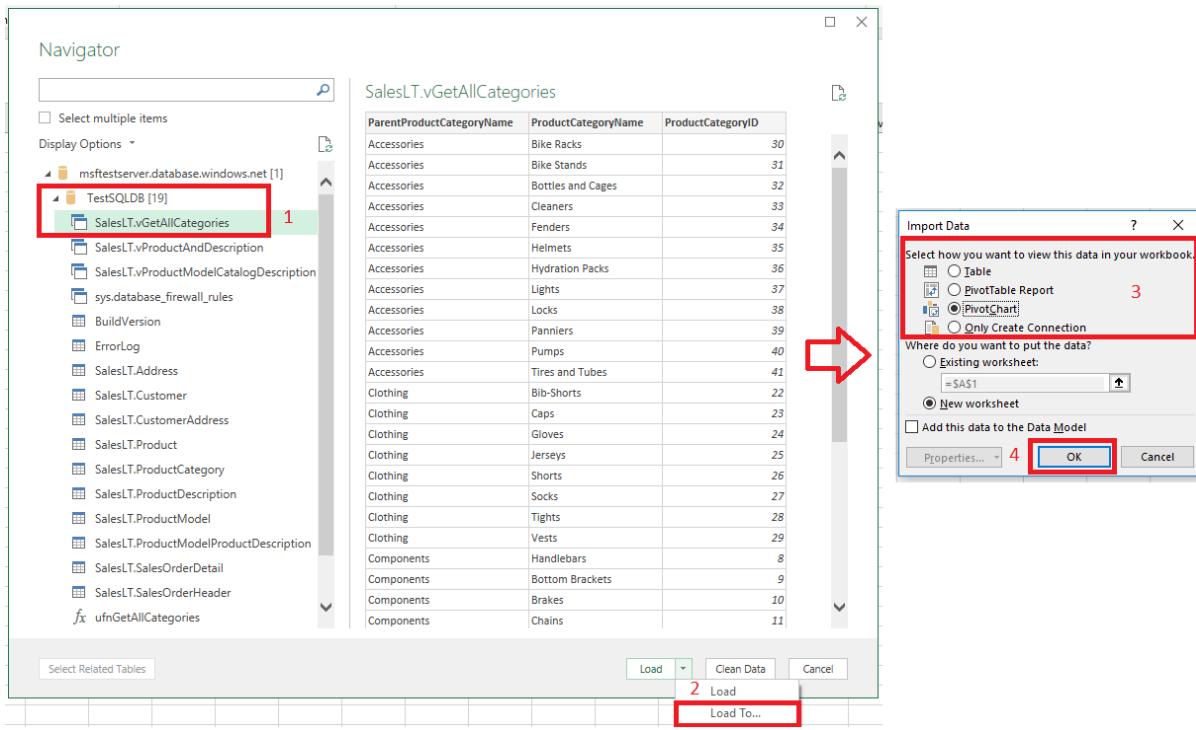
5. In the **Navigator**, select the database you want to work with from the list, select the tables or views you want to work with (we chose **vGetAllCategories**), and then select **Load** to move the data from your database to your Excel spreadsheet.

ParentProductCategoryName	ProductCategoryName	ProductCategoryID
Accessories	Bike Racks	30
Accessories	Bike Stands	31
Accessories	Bottles and Cages	32
Accessories	Cleaners	33
Accessories	Fenders	34
Accessories	Helmets	35
Accessories	Hydration Packs	36
Accessories	Lights	37
Accessories	Locks	38
Accessories	Panniers	39
Accessories	Pumps	40
Accessories	Tires and Tubes	41
Clothing	Bib-Shorts	22
Clothing	Caps	23
Clothing	Gloves	24
Clothing	Jerseys	25
Clothing	Shorts	26
Clothing	Socks	27
Clothing	Tights	28
Clothing	Vests	29
Components	Handlebars	8
Components	Bottom Brackets	9
Components	Brakes	10
Components	Chains	11

## Import the data into Excel and create a pivot chart

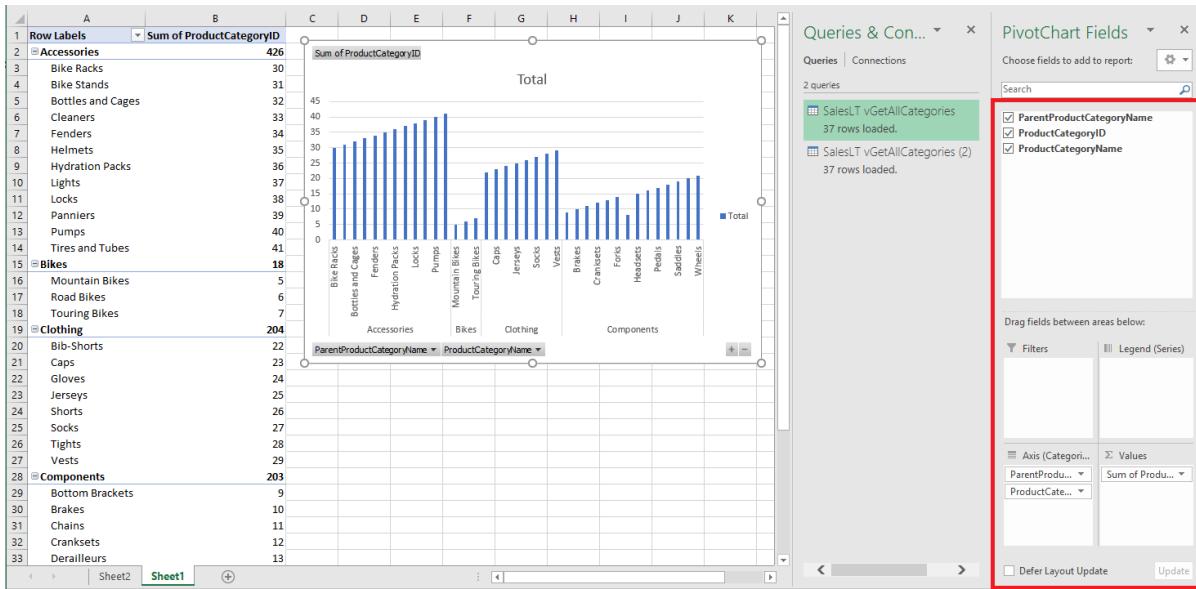
Now that you've established the connection, you have several different options with how to load the data. For example, the following steps create a pivot chart based on the data found in your SQL Database.

1. Follow the steps in the previous section, but this time, instead of selecting **Load**, select **Load to** from the **Load** drop-down.
2. Next, select how you want to view this data in your workbook. We chose **PivotChart**. You can also choose to create a **New worksheet** or to **Add this data to a Data Model**. For more information on Data Models, see [Create a data model in Excel](#).



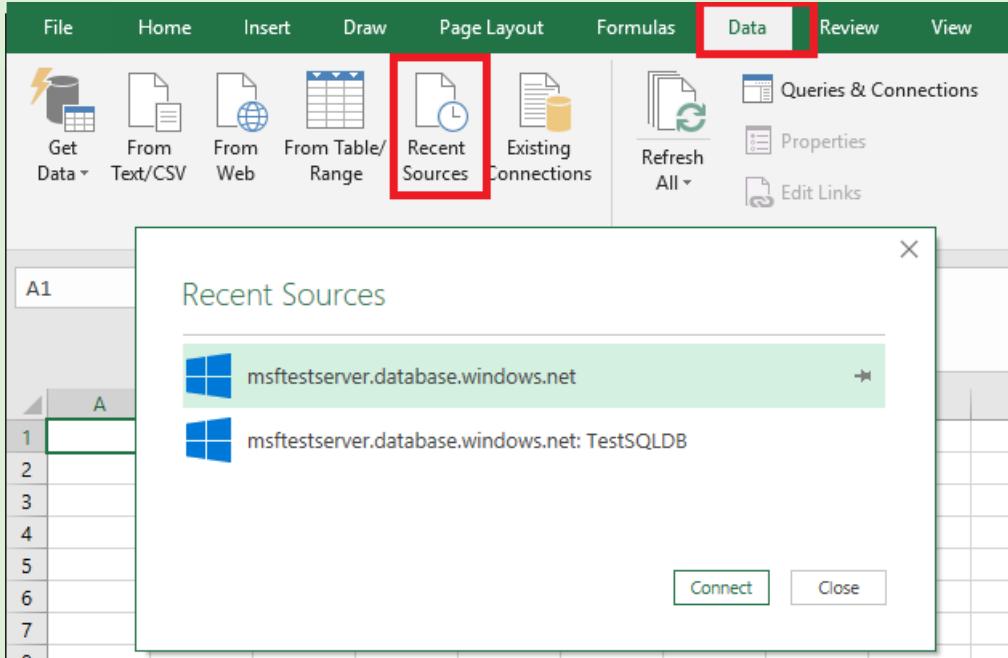
The worksheet now has an empty pivot table and chart.

### 3. Under **PivotTable Fields**, select all the check-boxes for the fields you want to view.



## TIP

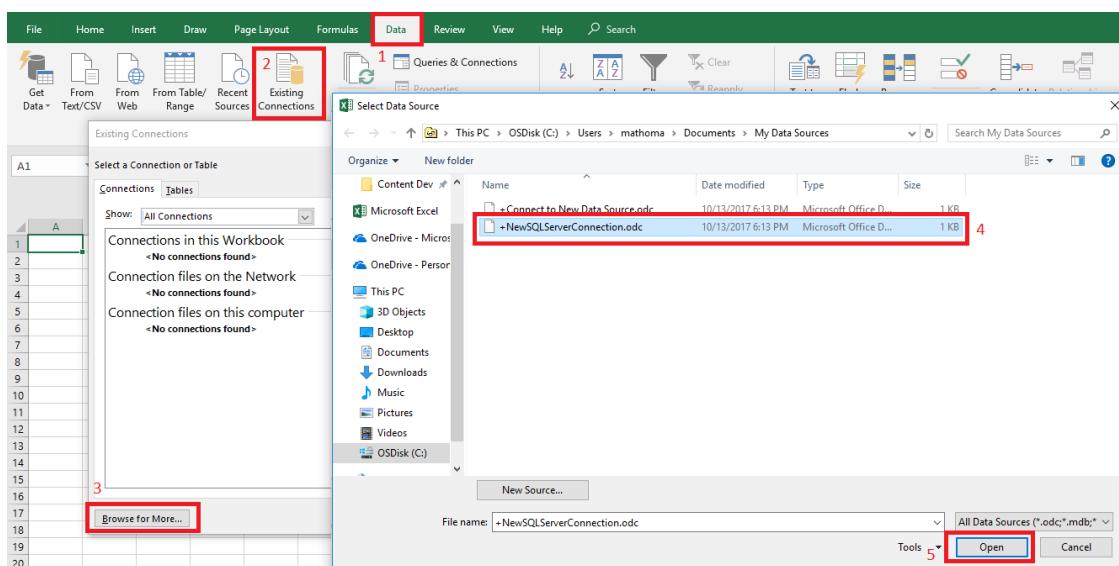
If you want to connect other Excel workbooks and worksheets to the database, select the **Data** tab, and select **Recent Sources** to launch the **Recent Sources** dialog box. From there, choose the connection you created from the list, and then



## Create a permanent connection using .odc file

To save the connection details permanently, you can create an .odc file and make this connection a selectable option within the **Existing Connections** dialog box.

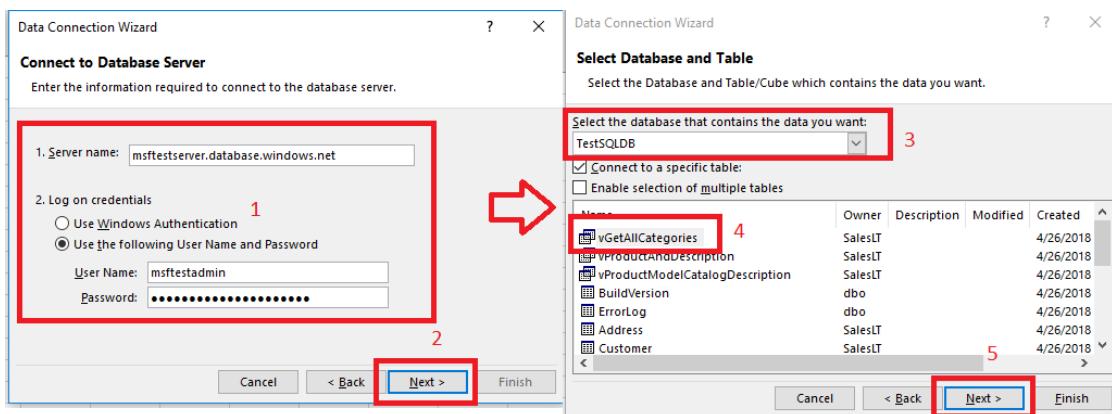
1. In the menu bar at the top of the page, select the **Data** tab, and then select **Existing Connections** to launch the **Existing Connections** dialog box.
  - a. Select **Browse for more** to open the **Select Data Source** dialog box.
  - b. Select the **+NewSqlServerConnection.odc** file and then select **Open** to open the **Data Connection Wizard**.



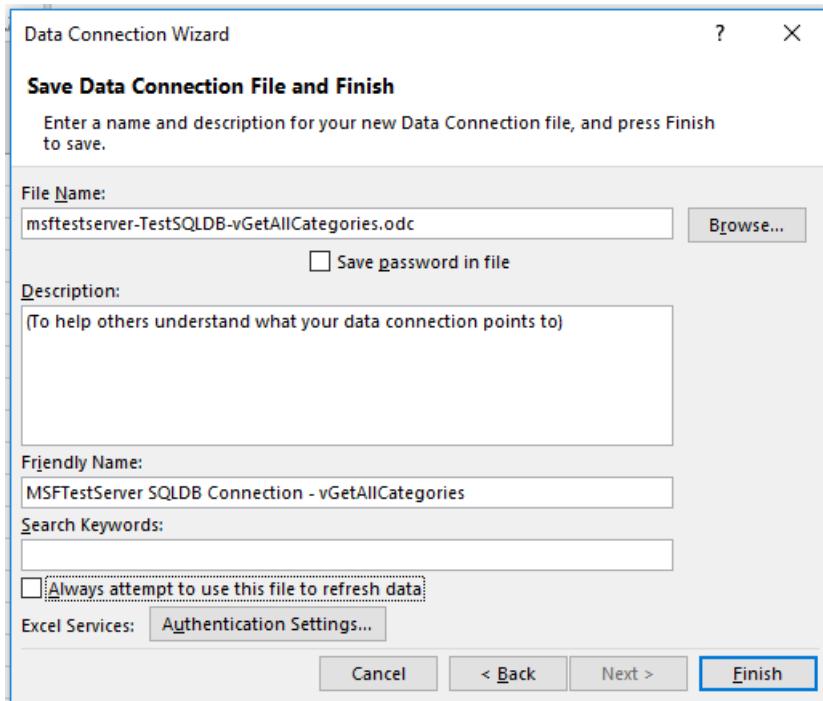
2. In the **Data Connection Wizard**, type in your server name and your SQL Database credentials. Select **Next**.
  - a. Select the database that contains your data from the drop-down.

b. Select the table or view you're interested in. We chose vGetAllCategories.

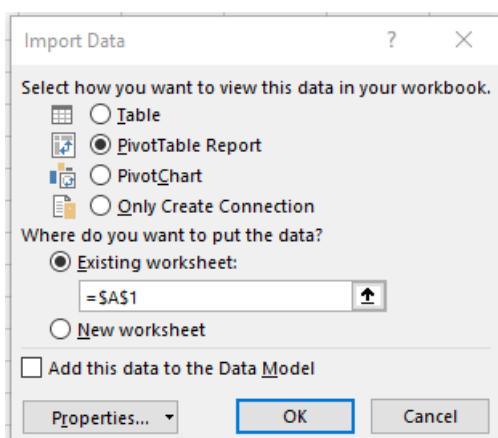
c. Select **Next**.



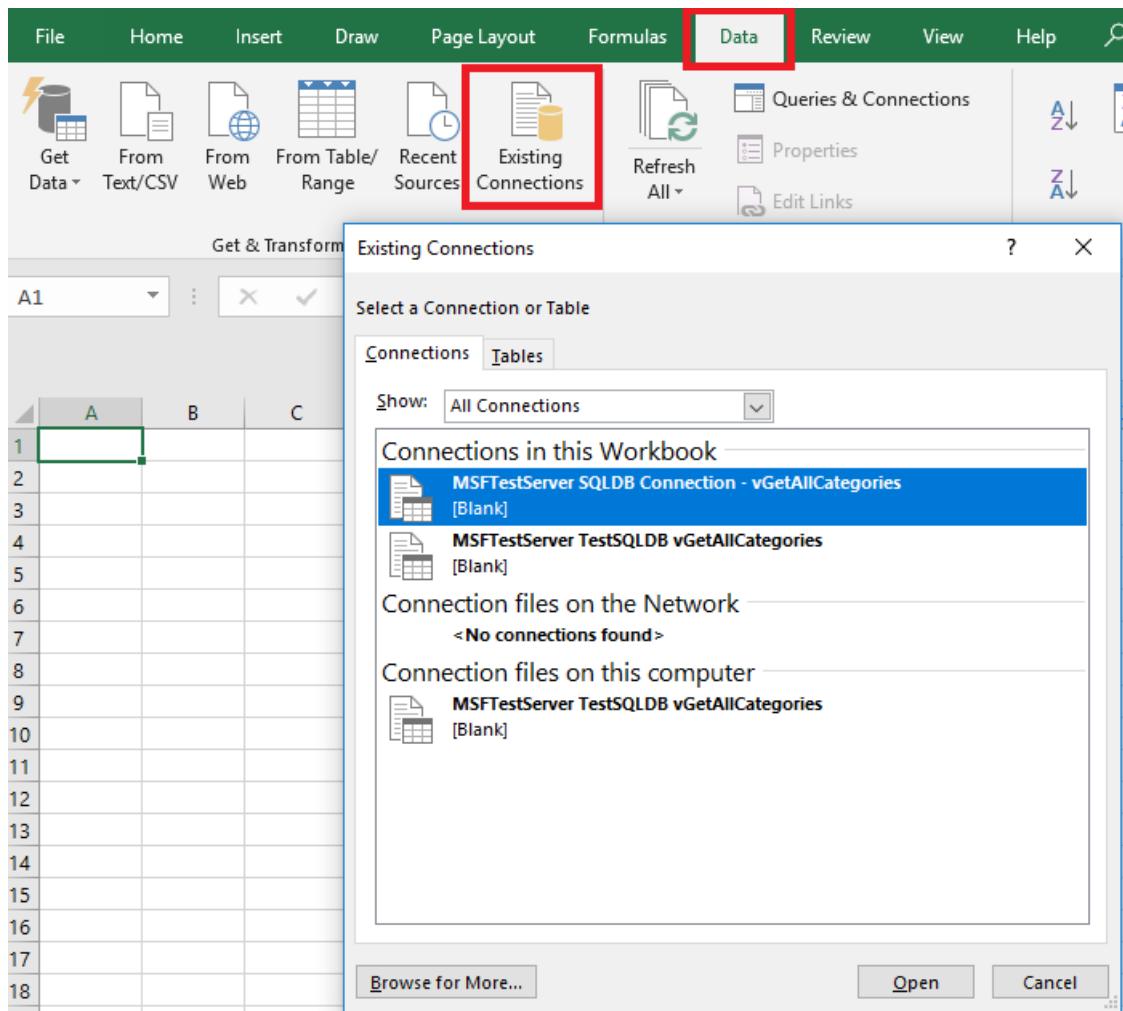
3. Select the location of your file, the **File Name**, and the **Friendly Name** in the next screen of the Data Connection Wizard. You can also choose to save the password in the file, though this can potentially expose your data to unwanted access. Select **Finish** when ready.



4. Select how you want to import your data. We chose to do a PivotTable. You can also modify the properties of the connection by select **Properties**. Select **OK** when ready. If you did not choose to save the password with the file, then you will be prompted to enter your credentials.



5. Verify that your new connection has been saved by expanding the **Data** tab, and selecting **Existing Connections**.



## Next steps

- Learn how to [Connect to SQL Database with SQL Server Management Studio](#) for advanced querying and analysis.
- Learn about the benefits of [elastic pools](#).
- Learn how to [create a web application that connects to SQL Database on the back-end](#).

# Designing globally available services using Azure SQL Database

2/26/2020 • 11 minutes to read • [Edit Online](#)

When building and deploying cloud services with Azure SQL Database, you use [active geo-replication](#) or [auto-failover groups](#) to provide resilience to regional outages and catastrophic failures. The same feature allows you to create globally distributed applications optimized for local access to the data. This article discusses common application patterns, including the benefits and trade-offs of each option.

## NOTE

If you are using Premium or Business Critical databases and elastic pools, you can make them resilient to regional outages by converting them to zone redundant deployment configuration. See [Zone-redundant databases](#).

## Scenario 1: Using two Azure regions for business continuity with minimal downtime

In this scenario, the applications have the following characteristics:

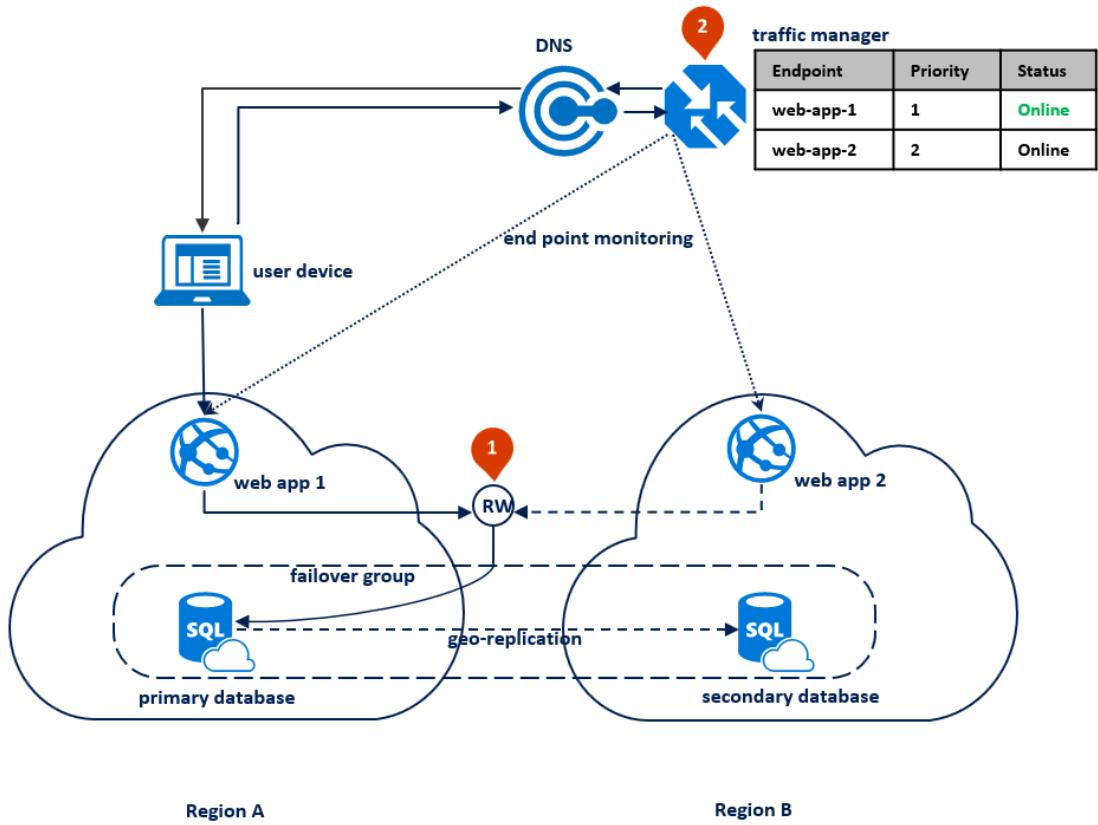
- Application is active in one Azure region
- All database sessions require read and write access (RW) to data
- Web tier and data tier must be collocated to reduce latency and traffic cost
- Fundamentally, downtime is a higher business risk for these applications than data loss

In this case, the application deployment topology is optimized for handling regional disasters when all application components need to failover together. The diagram below shows this topology. For geographic redundancy, the application's resources are deployed to Region A and B. However, the resources in Region B are not utilized until Region A fails. A failover group is configured between the two regions to manage database connectivity, replication and failover. The web service in both regions is configured to access the database via the read-write listener `<failover-group-name>.database.windows.net` (1). Traffic manager is set up to use [priority routing method](#) (2).

## NOTE

[Azure traffic manager](#) is used throughout this article for illustration purposes only. You can use any load-balancing solution that supports priority routing method.

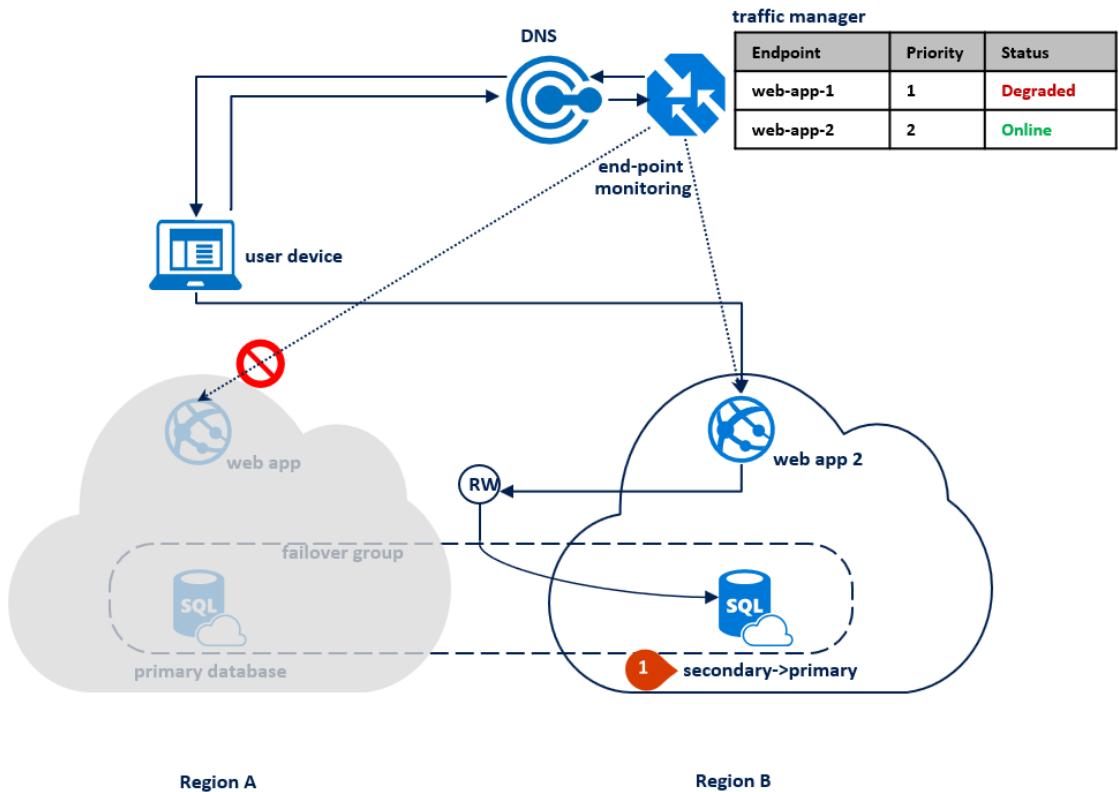
The following diagram shows this configuration before an outage:



After an outage in the primary region, the SQL Database service detects that the primary database is not accessible and triggers failover to the secondary region based on the parameters of the automatic failover policy (1). Depending on your application SLA, you can configure a grace period that controls the time between the detection of the outage and the failover itself. It is possible that traffic manager initiates the endpoint failover before the failover group triggers the failover of the database. In that case the web application cannot immediately reconnect to the database. But the reconnections will automatically succeed as soon as the database failover completes. When the failed region is restored and back online, the old primary automatically reconnects as a new secondary. The diagram below illustrates the configuration after failover.

#### NOTE

All transactions committed after the failover are lost during the reconnection. After the failover is completed, the application in region B is able to reconnect and restart processing the user requests. Both the web application and the primary database are now in region B and remain co-located.

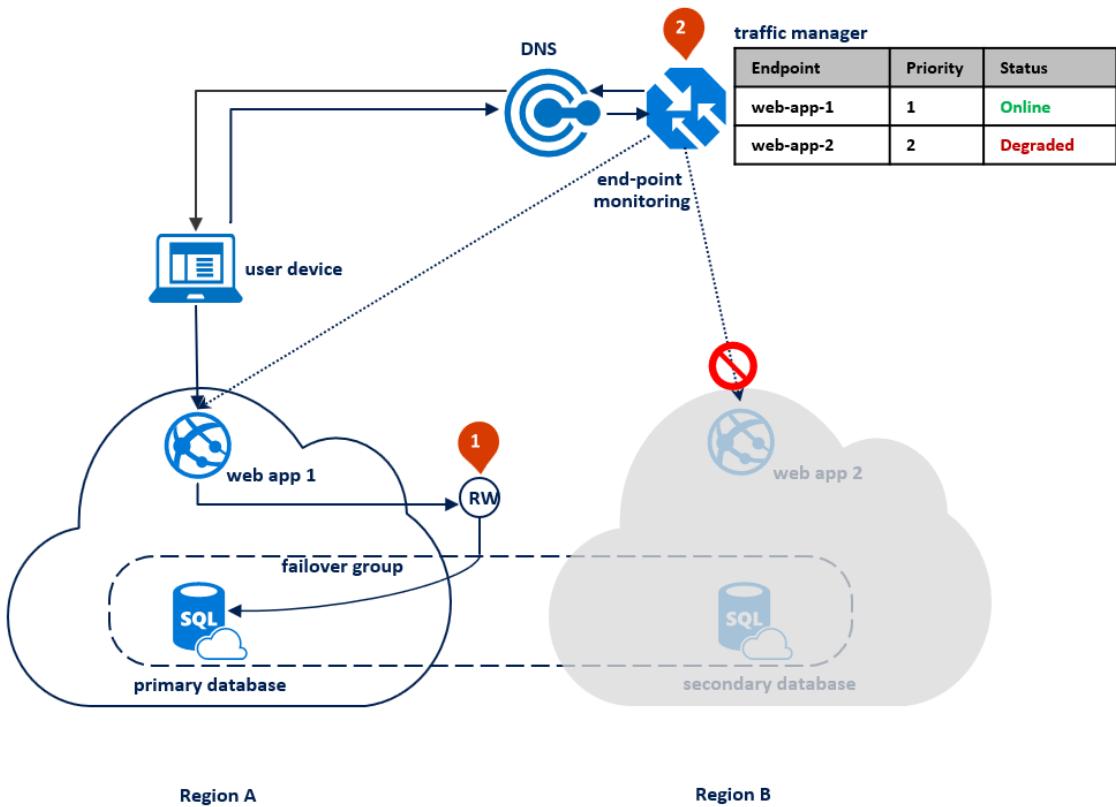


If an outage happens in region B, the replication process between the primary and the secondary database gets suspended but the link between the two remains intact (1). Traffic managed detects that connectivity to Region B is broken and marks the endpoint web app 2 as Degraded (2). The application's performance is not impacted in this case, but the database becomes exposed and therefore at higher risk of data loss in case region A fails in succession.

#### **NOTE**

For disaster recovery, we recommend the configuration with application deployment limited to two regions. This is because most of the Azure geographies have only two regions. This configuration does not protect your application from a simultaneous catastrophic failure of both regions. In an unlikely event of such a failure, you can recover your databases in a third region using [geo-restore operation](#).

Once the outage is mitigated, the secondary database automatically resynchronizes with the primary. During synchronization, performance of the primary can be impacted. The specific impact depends on the amount of data the new primary acquired since the failover. The following diagram illustrates an outage in the secondary region:



The key **advantages** of this design pattern are:

- The same web application is deployed to both regions without any region-specific configuration and doesn't require additional logic to manage failover.
- Application performance is not impacted by failover as the web application and the database are always co-located.

The main **tradeoff** is that the application resources in Region B are underutilized most of the time.

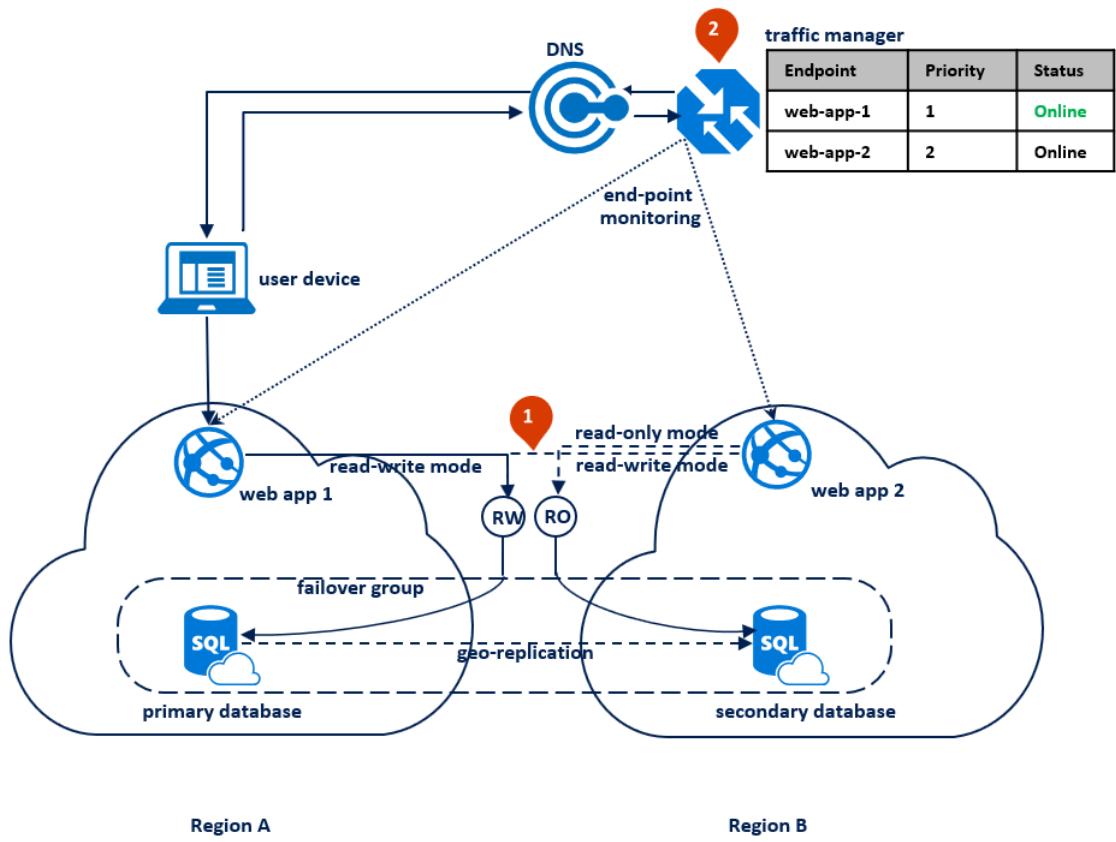
## Scenario 2: Azure regions for business continuity with maximum data preservation

This option is best suited for applications with the following characteristics:

- Any data loss is high business risk. The database failover can only be used as a last resort if the outage is caused by a catastrophic failure.
- The application supports read-only and read-write modes of operations and can operate in "read-only mode" for a period of time.

In this pattern, the application switches to read-only mode when the read-write connections start getting time-out errors. The Web Application is deployed to both regions and include a connection to the read-write listener endpoint and different connection to the read-only listener endpoint (1). The Traffic manager profile should use [priority routing](#). [End point monitoring](#) should be enabled for the application endpoint in each region (2).

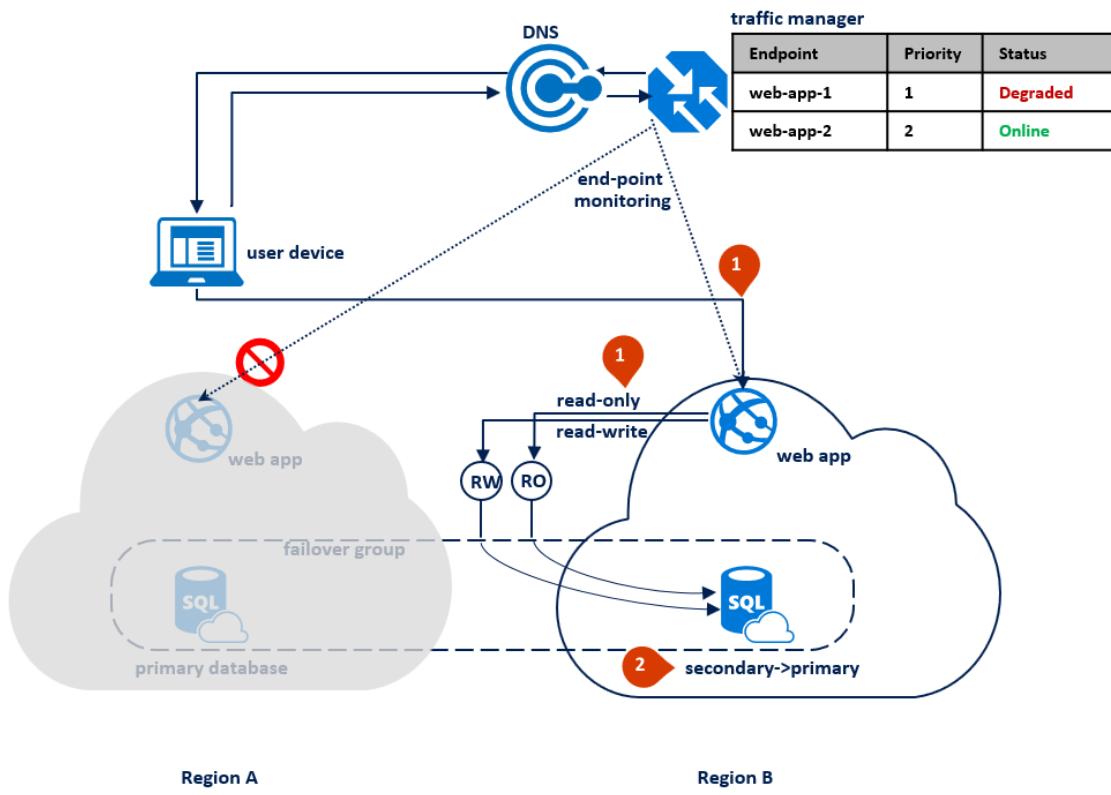
The following diagram illustrates this configuration before an outage:



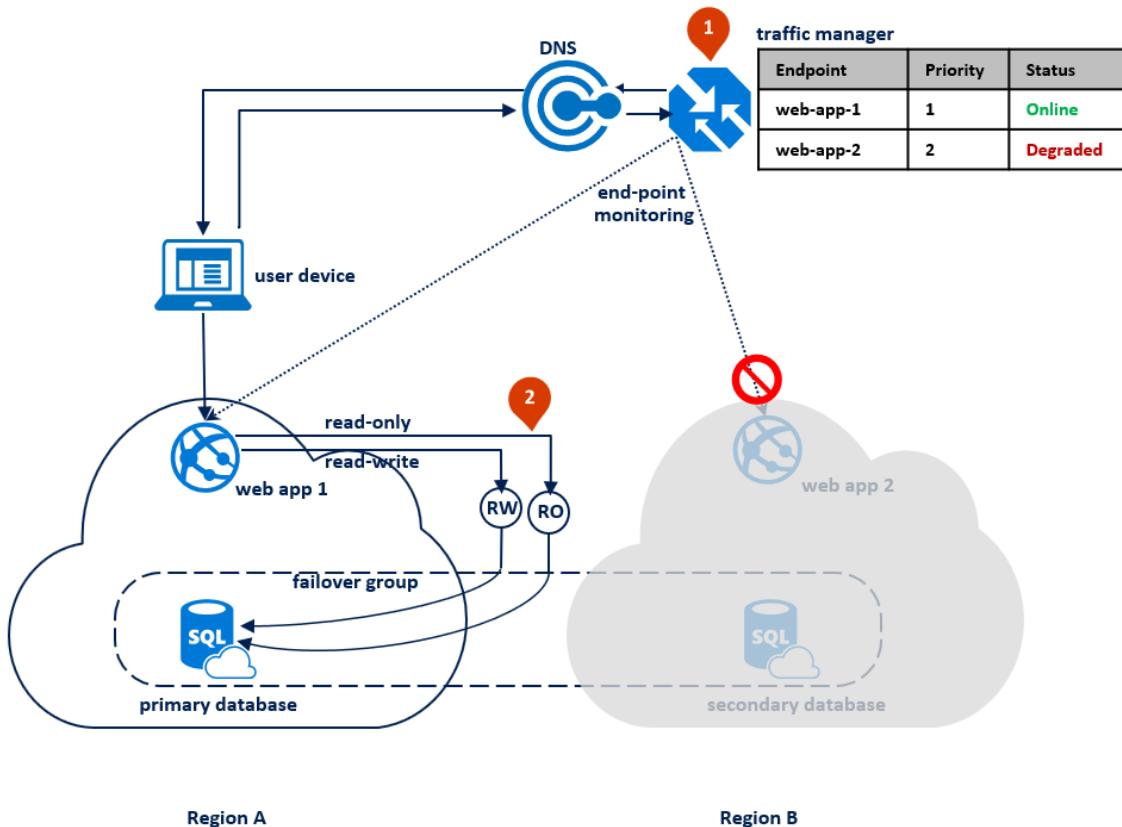
When the traffic manager detects a connectivity failure to region A, it automatically switches user traffic to the application instance in region B. With this pattern, it is important that you set the grace period with data loss to a sufficiently high value, for example 24 hours. It ensures that data loss is prevented if the outage is mitigated within that time. When the Web application in region B is activated the read-write operations start failing. At that point, it should switch to the read-only mode (1). In this mode the requests are automatically routed to the secondary database. If the outage is caused by a catastrophic failure, most likely it cannot be mitigated within the grace period. When it expires the failover group triggers the failover. After that the read-write listener becomes available and the connections to it stop failing (2). The following diagram illustrates the two stages of the recovery process.

#### NOTE

If the outage in the primary region is mitigated within the grace period, traffic manager detects the restoration of connectivity in the primary region and switches user traffic back to the application instance in region A. That application instance resumes and operates in read-write mode using the primary database in region A as illustrated by the previous diagram.



If an outage happens in region B, the traffic manager detects the failure of the end point web-app-2 in region B and marks it degraded (1). In the meantime, the failover group switches the read-only listener to region A (2). This outage does not impact the end user experience but the primary database is exposed during the outage. The following diagram illustrates a failure in the secondary region:



Once the outage is mitigated, the secondary database is immediately synchronized with the primary and the read-only listener is switched back to the secondary database in region B. During synchronization performance of the primary could be slightly impacted depending on the amount of data that needs to be synchronized.

This design pattern has several **advantages**:

- It avoids data loss during the temporary outages.
- Downtime depends only on how quickly traffic manager detects the connectivity failure, which is configurable.

The **tradeoff** is that the application must be able to operate in read-only mode.

## Scenario 3: Application relocation to a different geography without data loss and near zero downtime

In this scenario the application has the following characteristics:

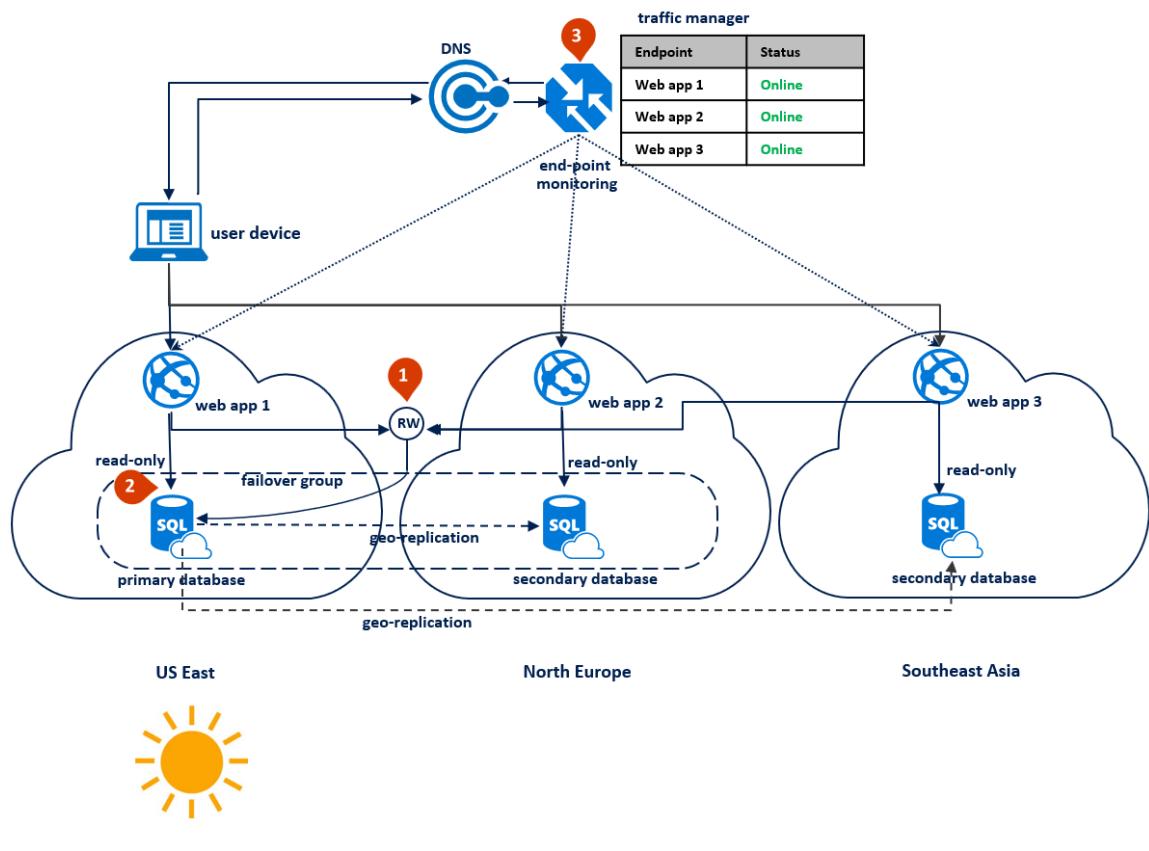
- The end users access the application from different geographies
- The application includes read-only workloads that do not depend on full synchronization with the latest updates
- Write access to data should be supported in the same geography for majority of the users
- Read latency is critical for the end user experience

In order to meet these requirements you need to guarantee that the user device **always** connects to the application deployed in the same geography for the read-only operations, such as browsing data, analytics etc. Whereas, the OLTP operations are processed in the same geography **most of the time**. For example, during the day time OLTP operations are processed in the same geography, but during the off hours they could be processed in a different geography. If the end user activity mostly happens during the working hours, you can guarantee the optimal performance for most of the users most of the time. The following diagram shows this topology.

The application's resources should be deployed in each geography where you have substantial usage demand. For example, if your application is actively used in the United States, European Union and South East Asia the application should be deployed to all of these geographies. The primary database should be dynamically switched from one geography to the next at the end of the working hours. This method is called "follow the sun". The OLTP workload always connects to the database via the read-write listener **<failover-group-name>.database.windows.net** (1). The read-only workload connects to the local database directly using the databases server endpoint **<server-name>.database.windows.net** (2). Traffic manager is configured with the **performance routing method**. It ensures that the end user's device is connected to the web service in the closest region. Traffic manager should be set up with end point monitoring enabled for each web service end point (3).

### NOTE

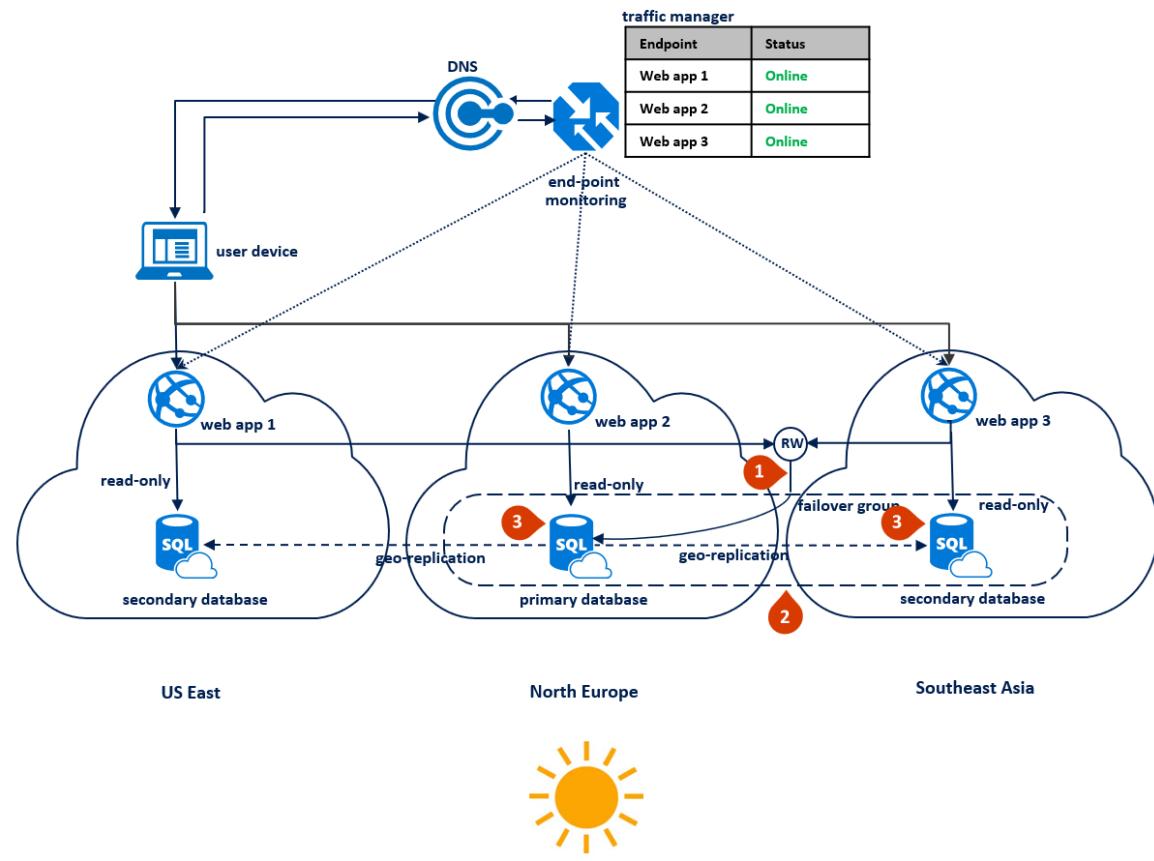
The failover group configuration defines which region is used for failover. Because the new primary is in a different geography the failover results in longer latency for both OLTP and read-only workloads until the impacted region is back online.



At the end of the day, for example at 11 PM local time, the active databases should be switched to the next region (North Europe). This task can be fully automated by using [Azure Logic Apps](#). The task involves the following steps:

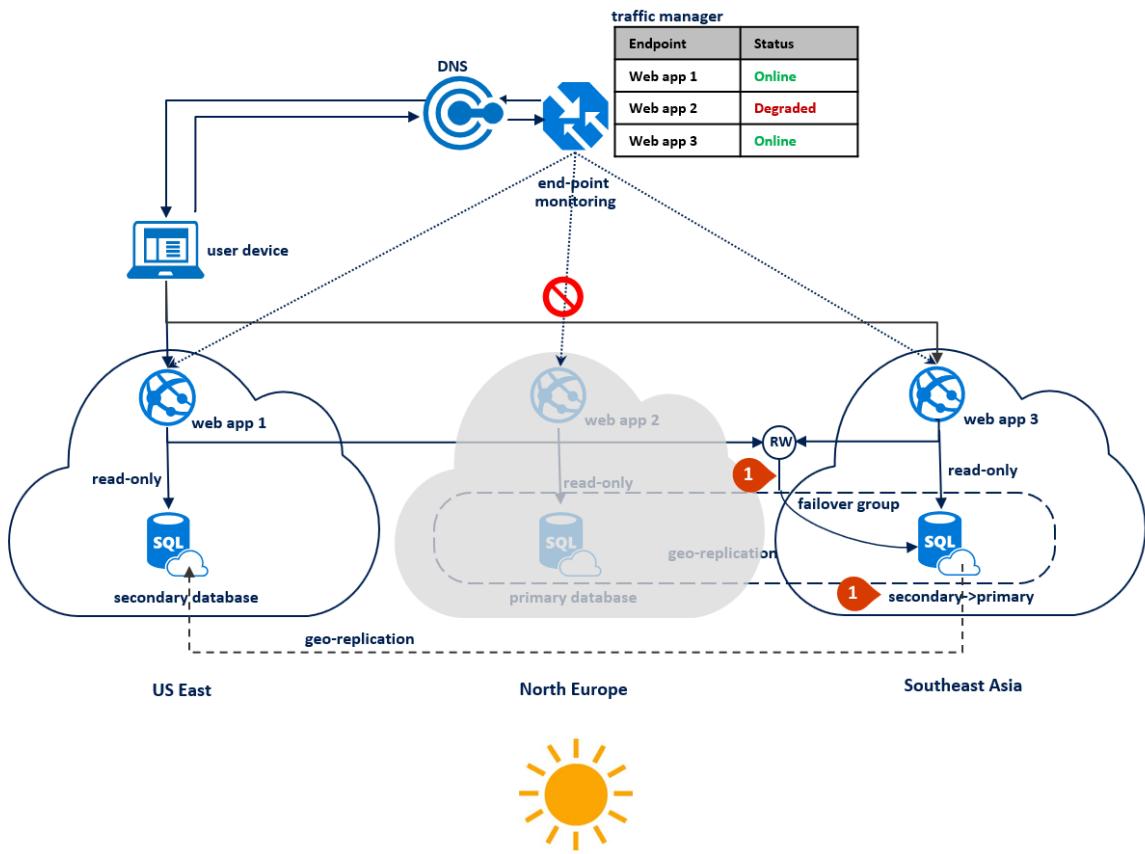
- Switch primary server in the failover group to North Europe using friendly failover (1)
- Remove the failover group between East US and North Europe
- Create a new failover group with the same name but between North Europe and East Asia (2).
- Add the primary in North Europe and secondary in East Asia to this failover group (3).

The following diagram illustrates the new configuration after the planned failover:



If an outage happens in North Europe for example, the automatic database failover is initiated by the failover group, which effectively results in moving the application to the next region ahead of schedule (1). In that case the US East is the only remaining secondary region until North Europe is back online. The remaining two regions serve the customers in all three geographies by switching roles. Azure Logic Apps has to be adjusted accordingly. Because the remaining regions get additional user traffic from Europe, the application's performance is impacted not only by additional latency but also by an increased number of end user connections. Once the outage is mitigated in North Europe, the secondary database there is immediately synchronized with the current primary. The following diagram illustrates an outage in North Europe:





#### NOTE

You can reduce the time when the end user's experience in Europe is degraded by the long latency. To do that you should proactively deploy an application copy and create the secondary database(s) in another local region (West Europe) as a replacement of the offline application instance in North Europe. When the latter is back online you can decide whether to continue using West Europe or to remove the copy of the application there and switch back to using North Europe.

The key **benefits** of this design are:

- The read-only application workload accesses data in the closest region at all times.
- The read-write application workload accesses data in the closest region during the period of the highest activity in each geography
- Because the application is deployed to multiple regions, it can survive a loss of one of the regions without any significant downtime.

But there are some **tradeoffs**:

- A regional outage results in the geography to be impacted by longer latency. Both read-write and read-only workloads are served by the application in a different geography.
- The read-only workloads must connect to a different endpoint in each region.

## Business continuity planning: Choose an application design for cloud disaster recovery

Your specific cloud disaster recovery strategy can combine or extend these design patterns to best meet the needs of your application. As mentioned earlier, the strategy you choose is based on the SLA you want to offer to your customers and the application deployment topology. To help guide your decision, the following table compares the choices based on recovery point objective (RPO) and estimated recovery time (ERT).

PATTERN	RPO	ERT
Active-passive deployment for disaster recovery with co-located database access	Read-write access < 5 sec	Failure detection time + DNS TTL
Active-active deployment for application load balancing	Read-write access < 5 sec	Failure detection time + DNS TTL
Active-passive deployment for data preservation	Read-only access < 5 sec	Read-only access = 0
	Read-write access = zero	Read-write access = Failure detection time + grace period with data loss

## Next steps

- For a business continuity overview and scenarios, see [Business continuity overview](#)
- To learn about active geo-replication, see [Active geo-replication](#).
- To learn about auto-failover groups, see [Auto-failover groups](#).
- For information about active geo-replication with elastic pools, see [Elastic pool disaster recovery strategies](#).

# Disaster recovery strategies for applications using SQL Database elastic pools

11/7/2019 • 13 minutes to read • [Edit Online](#)

Over the years we have learned that cloud services are not foolproof and catastrophic incidents happen. SQL Database provides several capabilities to provide for the business continuity of your application when these incidents occur. [Elastic pools](#) and single databases support the same kind of disaster recovery (DR) capabilities. This article describes several DR strategies for elastic pools that leverage these SQL Database business continuity features.

This article uses the following canonical SaaS ISV application pattern:

A modern cloud-based web application provisions one SQL database for each end user. The ISV has many customers and therefore uses many databases, known as tenant databases. Because the tenant databases typically have unpredictable activity patterns, the ISV uses an elastic pool to make the database cost very predictable over extended periods of time. The elastic pool also simplifies the performance management when the user activity spikes. In addition to the tenant databases the application also uses several databases to manage user profiles, security, collect usage patterns etc. Availability of the individual tenants does not impact the application's availability as whole. However, the availability and performance of management databases is critical for the application's function and if the management databases are offline the entire application is offline.

This article discusses DR strategies covering a range of scenarios from cost sensitive startup applications to ones with stringent availability requirements.

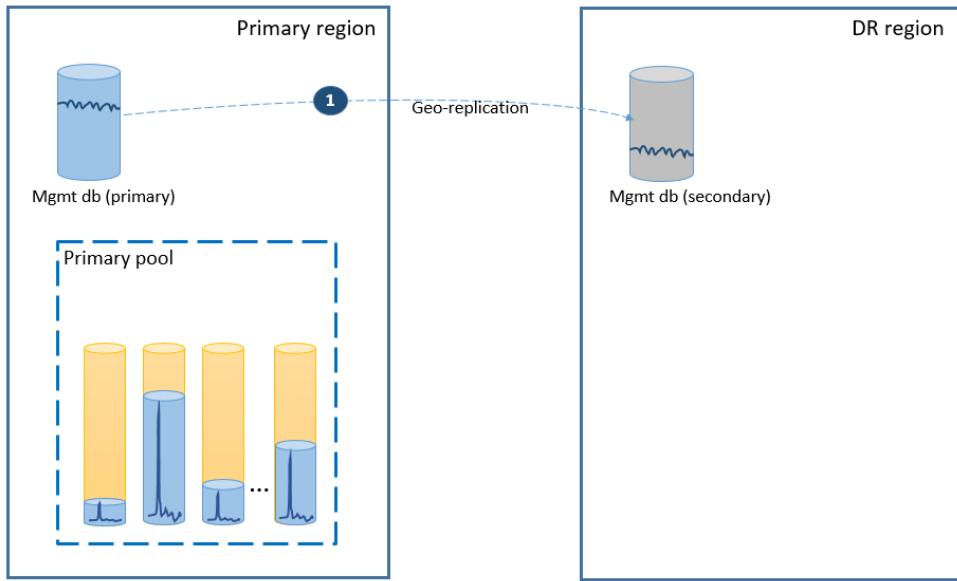
## NOTE

If you are using Premium or Business Critical databases and elastic pools, you can make them resilient to regional outages by converting them to zone redundant deployment configuration. See [Zone-redundant databases](#).

## Scenario 1. Cost sensitive startup

I am a startup business and am extremely cost sensitive. I want to simplify deployment and management of the application and I can have a limited SLA for individual customers. But I want to ensure the application as a whole is never offline.

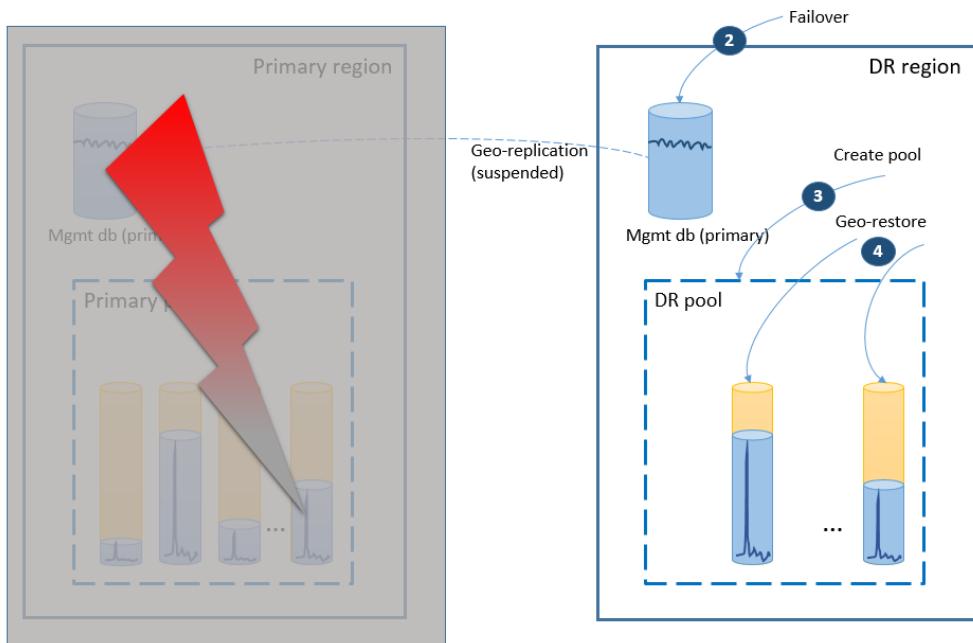
To satisfy the simplicity requirement, deploy all tenant databases into one elastic pool in the Azure region of your choice and deploy management databases as geo-replicated single databases. For the disaster recovery of tenants, use geo-restore, which comes at no additional cost. To ensure the availability of the management databases, geo-replicate them to another region using an auto-failover group (step 1). The ongoing cost of the disaster recovery configuration in this scenario is equal to the total cost of the secondary databases. This configuration is illustrated on the next diagram.



If an outage occurs in the primary region, the recovery steps to bring your application online are illustrated by the next diagram.

- The failover group initiates automatic failover of the management database to the DR region. The application is automatically reconnected to the new primary and all new accounts and tenant databases are created in the DR region. The existing customers see their data temporarily unavailable.
- Create the elastic pool with the same configuration as the original pool (2).
- Use geo-restore to create copies of the tenant databases (3). You can consider triggering the individual restores by the end-user connections or use some other application-specific priority scheme.

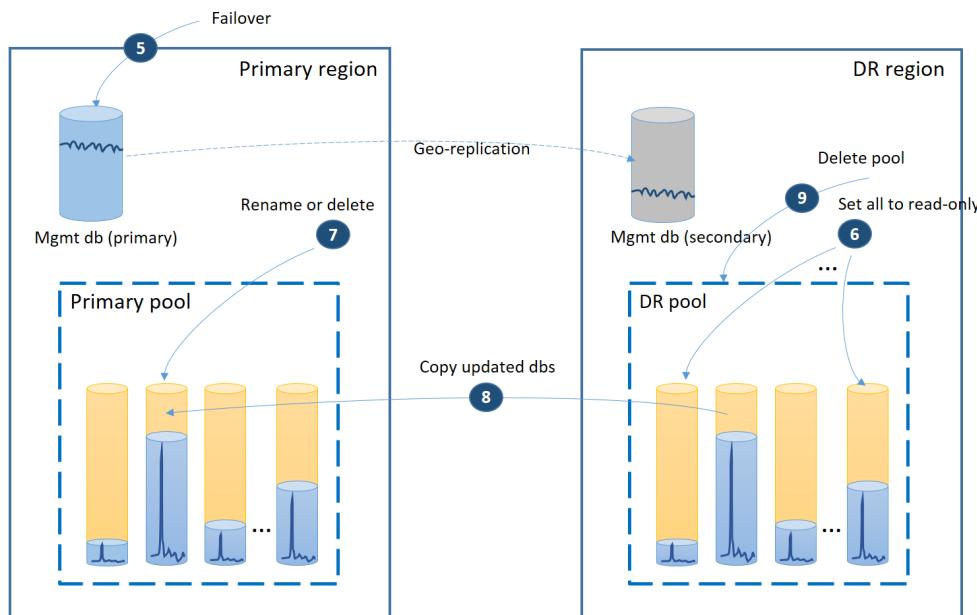
At this point your application is back online in the DR region, but some customers experience delay when accessing their data.



If the outage was temporary, it is possible that the primary region is recovered by Azure before all the database restores are complete in the DR region. In this case, orchestrate moving the application back to the primary region. The process takes the steps illustrated on the next diagram.

- Cancel all outstanding geo-restore requests.
- Fail over the management databases to the primary region (5). After the region's recovery, the old primaries have automatically become secondaries. Now they switch roles again.
- Change the application's connection string to point back to the primary region. Now all new accounts and tenant databases are created in the primary region. Some existing customers see their data temporarily unavailable.
- Set all databases in the DR pool to read-only to ensure they cannot be modified in the DR region (6).
- For each database in the DR pool that has changed since the recovery, rename or delete the corresponding databases in the primary pool (7).
- Copy the updated databases from the DR pool to the primary pool (8).
- Delete the DR pool (9)

At this point your application is online in the primary region with all tenant databases available in the primary pool.



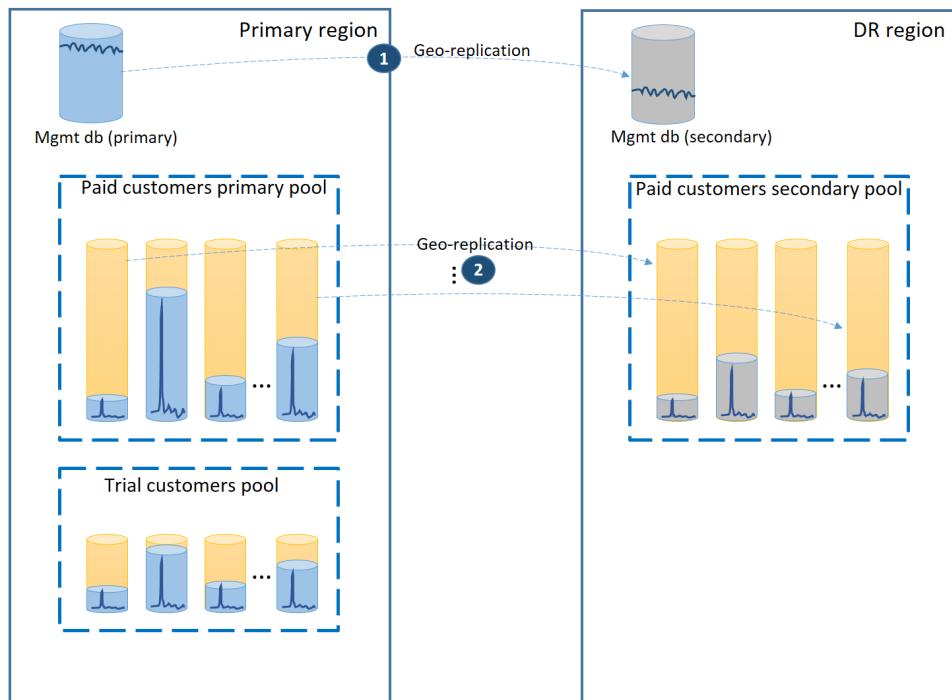
The key **benefit** of this strategy is low ongoing cost for data tier redundancy. Backups are taken automatically by the SQL Database service with no application rewrite and at no additional cost. The cost is incurred only when the elastic databases are restored. The **trade-off** is that the complete recovery of all tenant databases takes significant time. The length of time depends on the total number of restores you initiate in the DR region and overall size of the tenant databases. Even if you prioritize some tenants' restores over others, you are competing with all the other restores that are initiated in the same region as the service arbitrates and throttles to minimize the overall impact on the existing customers' databases. In addition, the recovery of the tenant databases cannot start until the new elastic pool in the DR region is created.

## Scenario 2. Mature application with tiered service

I am a mature SaaS application with tiered service offers and different SLAs for trial customers and for paying customers. For the trial customers, I have to reduce the cost as much as possible. Trial customers can take downtime but I want to reduce its likelihood. For the paying customers, any downtime is a flight risk. So I want to make sure that paying customers are always able to access their data.

To support this scenario, separate the trial tenants from paid tenants by putting them into separate elastic pools. The trial customers have lower eDTU or vCores per tenant and lower SLA with a longer recovery time. The paying customers are in a pool with higher eDTU or vCores per tenant and a higher SLA. To guarantee the lowest

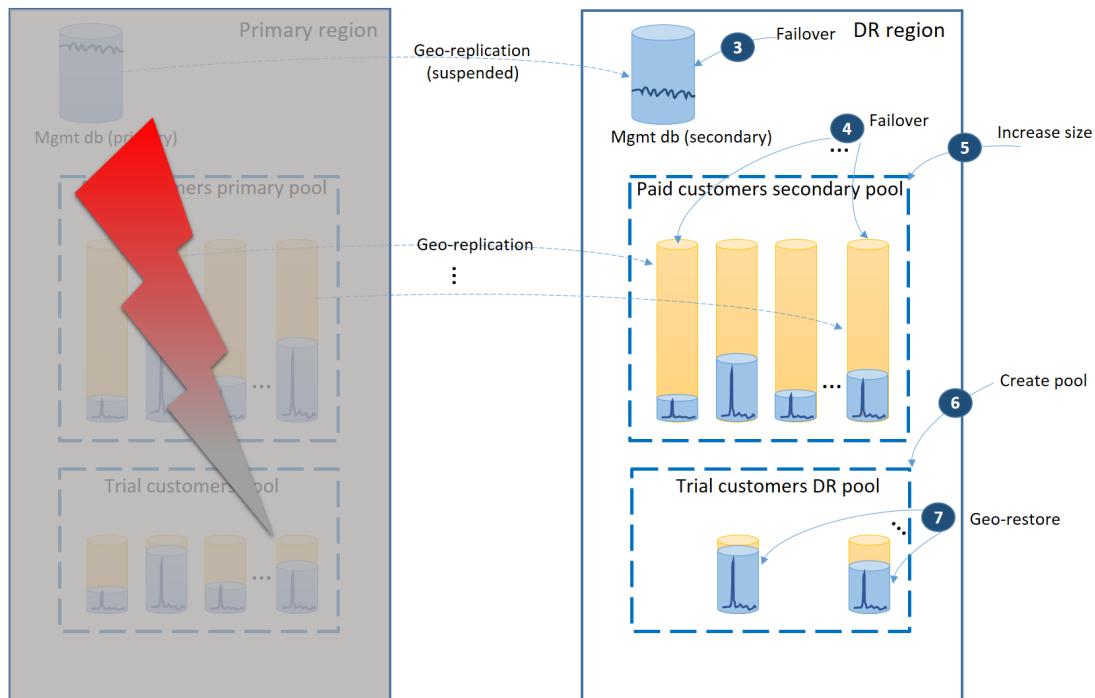
recovery time, the paying customers' tenant databases are geo-replicated. This configuration is illustrated on the next diagram.



As in the first scenario, the management databases are quite active so you use a single geo-replicated database for it (1). This ensures the predictable performance for new customer subscriptions, profile updates, and other management operations. The region in which the primaries of the management databases reside is the primary region and the region in which the secondaries of the management databases reside is the DR region.

The paying customers' tenant databases have active databases in the "paid" pool provisioned in the primary region. Provision a secondary pool with the same name in the DR region. Each tenant is geo-replicated to the secondary pool (2). This enables quick recovery of all tenant databases using failover.

If an outage occurs in the primary region, the recovery steps to bring your application online are illustrated in the next diagram:



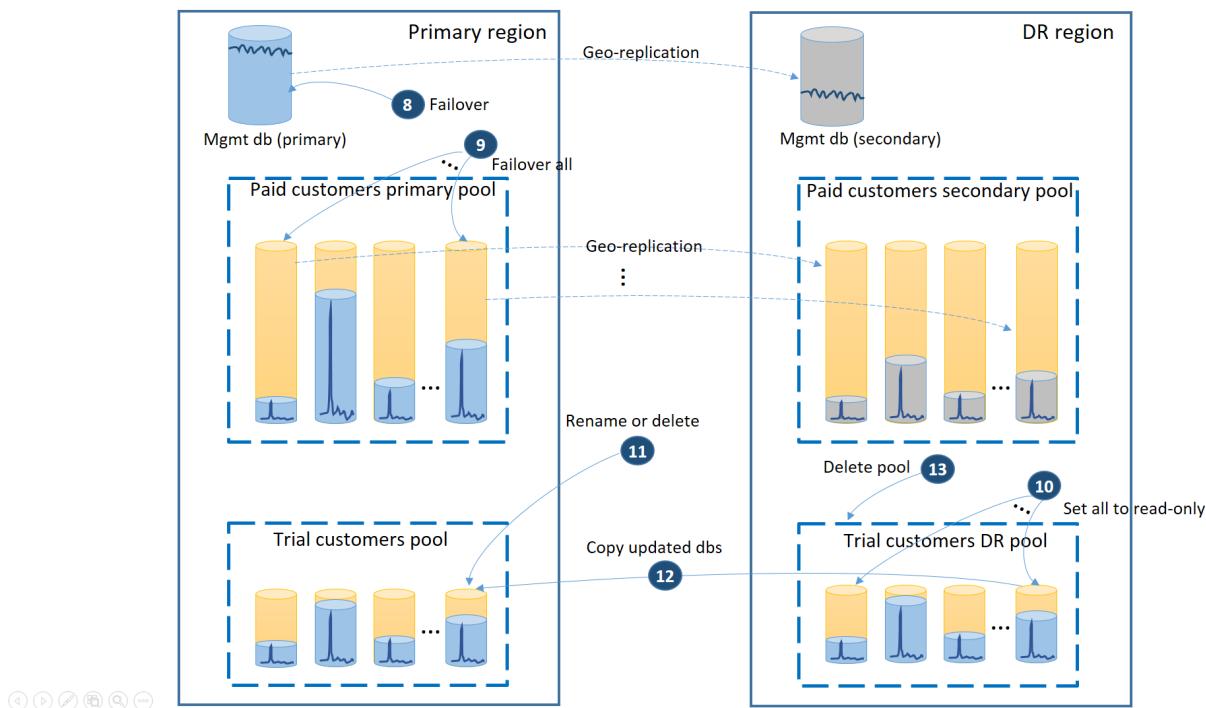
- Immediately fail over the management databases to the DR region (3).
- Change the application's connection string to point to the DR region. Now all new accounts and tenant

databases are created in the DR region. The existing trial customers see their data temporarily unavailable.

- Fail over the paid tenant's databases to the pool in the DR region to immediately restore their availability (4). Since the failover is a quick metadata level change, consider an optimization where the individual failovers are triggered on demand by the end-user connections.
- If your secondary pool eDTU size or vCore value was lower than the primary because the secondary databases only required the capacity to process the change logs while they were secondaries, immediately increase the pool capacity now to accommodate the full workload of all tenants (5).
- Create the new elastic pool with the same name and the same configuration in the DR region for the trial customers' databases (6).
- Once the trial customers' pool is created, use geo-restore to restore the individual trial tenant databases into the new pool (7). Consider triggering the individual restores by the end-user connections or use some other application-specific priority scheme.

At this point your application is back online in the DR region. All paying customers have access to their data while the trial customers experience delay when accessing their data.

When the primary region is recovered by Azure *after* you have restored the application in the DR region you can continue running the application in that region or you can decide to fail back to the primary region. If the primary region is recovered *before* the failover process is completed, consider failing back right away. The failback takes the steps illustrated in the next diagram:



- Cancel all outstanding geo-restore requests.
- Fail over the management databases (8). After the region's recovery, the old primary automatically become the secondary. Now it becomes the primary again.
- Fail over the paid tenant databases (9). Similarly, after the region's recovery, the old primaries automatically become the secondaries. Now they become the primaries again.
- Set the restored trial databases that have changed in the DR region to read-only (10).
- For each database in the trial customers DR pool that changed since the recovery, rename or delete the corresponding database in the trial customers primary pool (11).
- Copy the updated databases from the DR pool to the primary pool (12).
- Delete the DR pool (13).

#### NOTE

The failover operation is asynchronous. To minimize the recovery time it is important that you execute the tenant databases' failover command in batches of at least 20 databases.

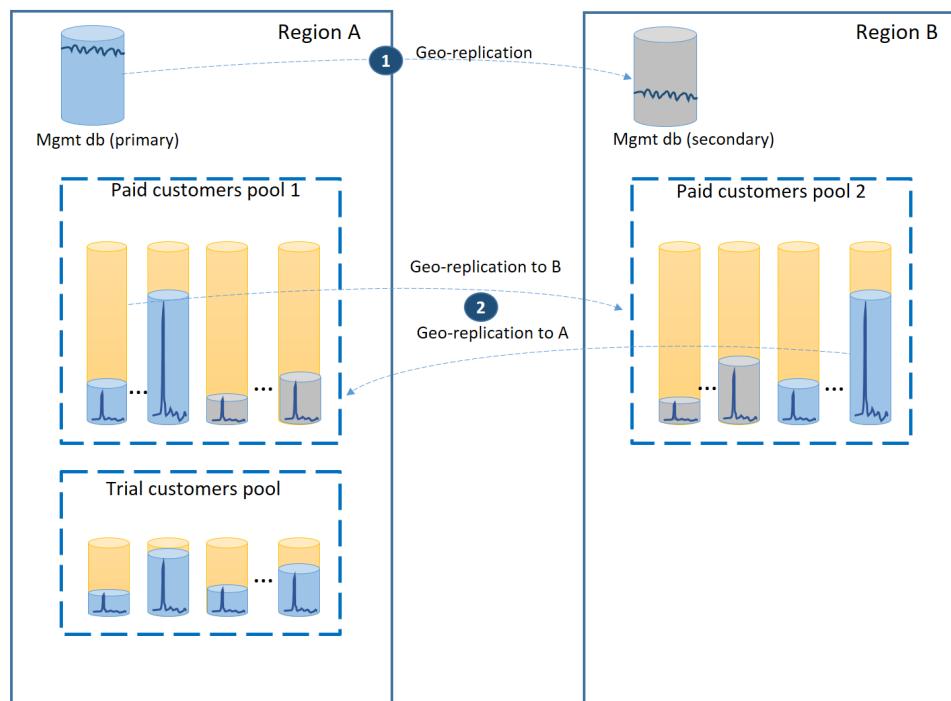
The key **benefit** of this strategy is that it provides the highest SLA for the paying customers. It also guarantees that the new trials are unblocked as soon as the trial DR pool is created. The **trade-off** is that this setup increases the total cost of the tenant databases by the cost of the secondary DR pool for paid customers. In addition, if the secondary pool has a different size, the paying customers experience lower performance after failover until the pool upgrade in the DR region is completed.

## Scenario 3. Geographically distributed application with tiered service

I have a mature SaaS application with tiered service offers. I want to offer a very aggressive SLA to my paid customers and minimize the risk of impact when outages occur because even brief interruption can cause customer dissatisfaction. It is critical that the paying customers can always access their data. The trials are free and an SLA is not offered during the trial period.

To support this scenario, use three separate elastic pools. Provision two equal size pools with high eDTUs or vCores per database in two different regions to contain the paid customers' tenant databases. The third pool containing the trial tenants can have lower eDTUs or vCores per database and be provisioned in one of the two regions.

To guarantee the lowest recovery time during outages, the paying customers' tenant databases are geo-replicated with 50% of the primary databases in each of the two regions. Similarly, each region has 50% of the secondary databases. This way, if a region is offline, only 50% of the paid customers' databases are impacted and have to fail over. The other databases remain intact. This configuration is illustrated in the following diagram:

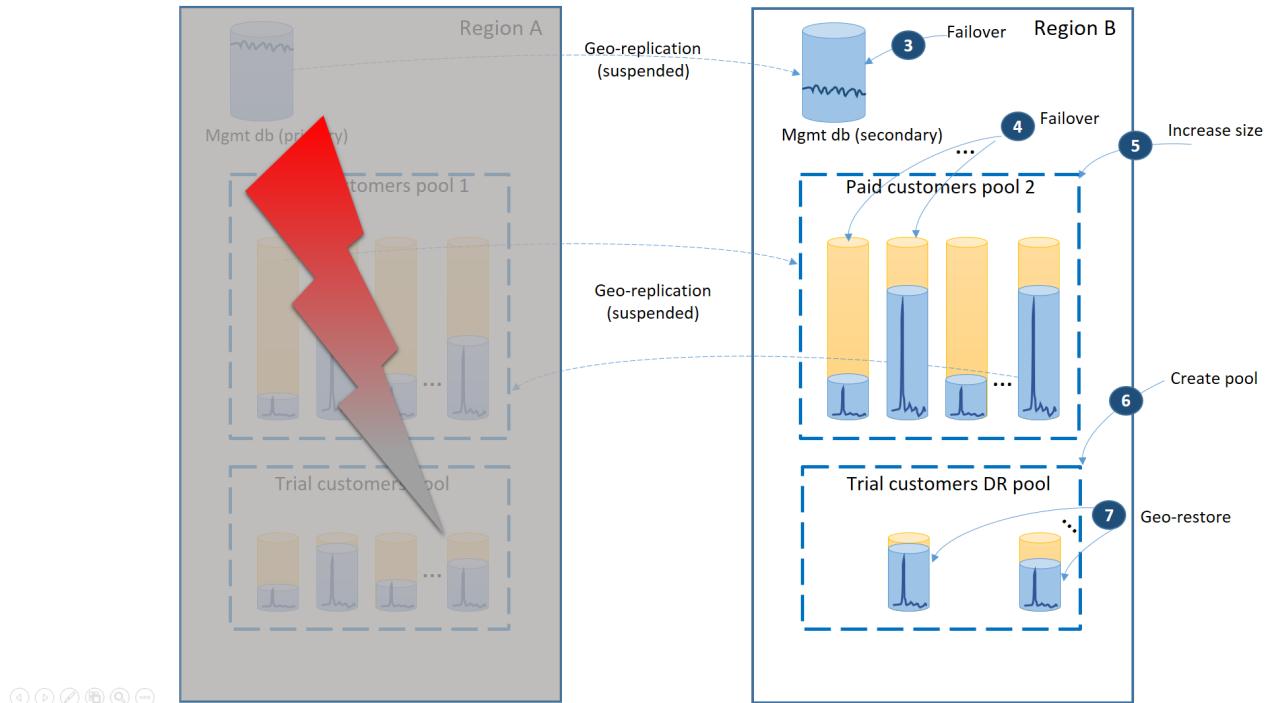


As in the previous scenarios, the management databases are quite active so configure them as single geo-replicated databases (1). This ensures the predictable performance of the new customer subscriptions, profile updates and other management operations. Region A is the primary region for the management databases and the region B is used for recovery of the management databases.

The paying customers' tenant databases are also geo-replicated but with primaries and secondaries split between region A and region B (2). This way, the tenant primary databases impacted by the outage can fail over to the

other region and become available. The other half of the tenant databases are not be impacted at all.

The next diagram illustrates the recovery steps to take if an outage occurs in region A.



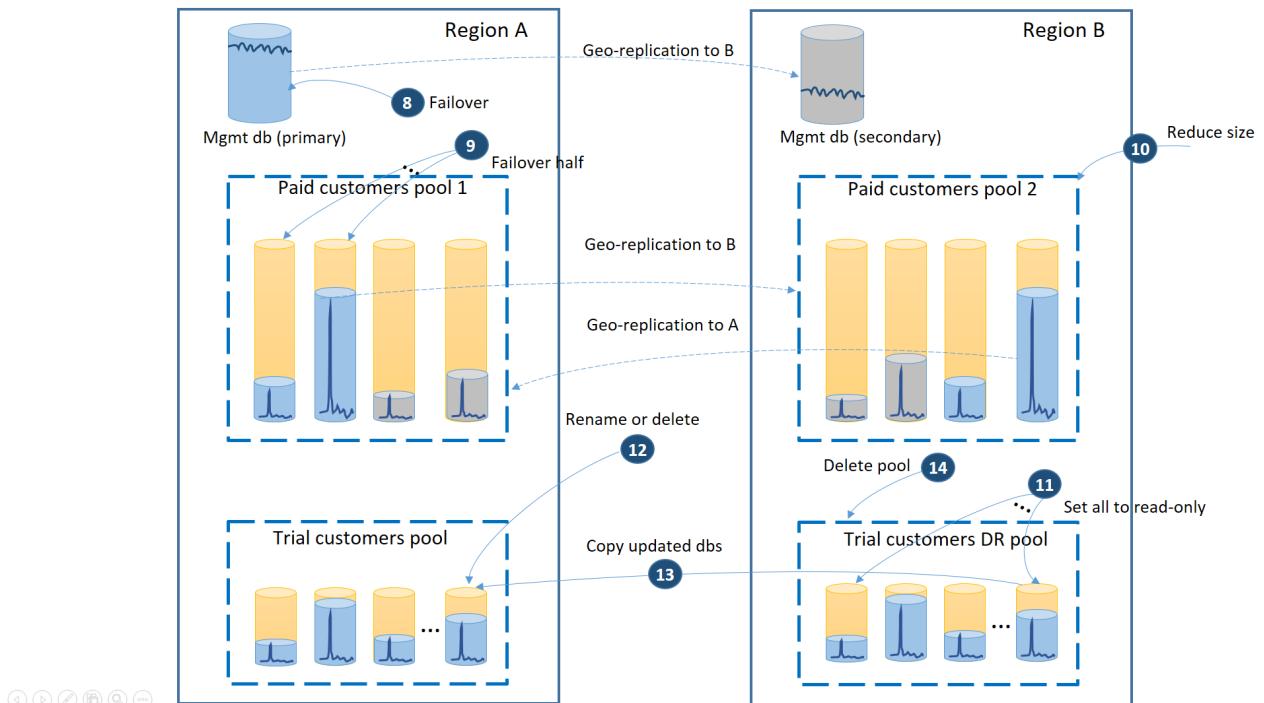
- Immediately fail over the management databases to region B (3).
- Change the application's connection string to point to the management databases in region B. Modify the management databases to make sure the new accounts and tenant databases are created in region B and the existing tenant databases are found there as well. The existing trial customers see their data temporarily unavailable.
- Fail over the paid tenant's databases to pool 2 in region B to immediately restore their availability (4). Since the failover is a quick metadata level change, you may consider an optimization where the individual failovers are triggered on demand by the end-user connections.
- Since now pool 2 contains only primary databases, the total workload in the pool increases and can immediately increase its eDTU size (5) or number of vCores.
- Create the new elastic pool with the same name and the same configuration in the region B for the trial customers' databases (6).
- Once the pool is created use geo-restore to restore the individual trial tenant database into the pool (7). You can consider triggering the individual restores by the end-user connections or use some other application-specific priority scheme.

#### NOTE

The failover operation is asynchronous. To minimize the recovery time, it is important that you execute the tenant databases' failover command in batches of at least 20 databases.

At this point your application is back online in region B. All paying customers have access to their data while the trial customers experience delay when accessing their data.

When region A is recovered you need to decide if you want to use region B for trial customers or fallback to using the trial customers pool in region A. One criteria could be the % of trial tenant databases modified since the recovery. Regardless of that decision, you need to re-balance the paid tenants between two pools. the next diagram illustrates the process when the trial tenant databases fail back to region A.



- Cancel all outstanding geo-restore requests to trial DR pool.
- Fail over the management database (8). After the region's recovery, the old primary automatically became the secondary. Now it becomes the primary again.
- Select which paid tenant databases fail back to pool 1 and initiate failover to their secondaries (9). After the region's recovery, all databases in pool 1 automatically became secondaries. Now 50% of them become primaries again.
- Reduce the size of pool 2 to the original eDTU (10) or number of vCores.
- Set all restored trial databases in the region B to read-only (11).
- For each database in the trial DR pool that has changed since the recovery, rename or delete the corresponding database in the trial primary pool (12).
- Copy the updated databases from the DR pool to the primary pool (13).
- Delete the DR pool (14).

The key **benefits** of this strategy are:

- It supports the most aggressive SLA for the paying customers because it ensures that an outage cannot impact more than 50% of the tenant databases.
- It guarantees that the new trials are unblocked as soon as the trial DR pool is created during the recovery.
- It allows more efficient use of the pool capacity as 50% of secondary databases in pool 1 and pool 2 are guaranteed to be less active than the primary databases.

The main **trade-offs** are:

- The CRUD operations against the management databases have lower latency for the end users connected to region A than for the end users connected to region B as they are executed against the primary of the management databases.
- It requires more complex design of the management database. For example, each tenant record has a location tag that needs to be changed during failover and fallback.
- The paying customers may experience lower performance than usual until the pool upgrade in region B is completed.

## Summary

This article focuses on the disaster recovery strategies for the database tier used by a SaaS ISV multi-tenant

application. The strategy you choose is based on the needs of the application, such as the business model, the SLA you want to offer to your customers, budget constraint etc. Each described strategy outlines the benefits and trade-off so you could make an informed decision. Also, your specific application likely includes other Azure components. So you review their business continuity guidance and orchestrate the recovery of the database tier with them. To learn more about managing recovery of database applications in Azure, refer to [Designing cloud solutions for disaster recovery](#).

## Next steps

- To learn about Azure SQL Database automated backups, see [SQL Database automated backups](#).
- For a business continuity overview and scenarios, see [Business continuity overview](#).
- To learn about using automated backups for recovery, see [restore a database from the service-initiated backups](#).
- To learn about faster recovery options, see [Active geo-replication](#) and [Auto-failover groups](#).
- To learn about using automated backups for archiving, see [database copy](#).

# Manage rolling upgrades of cloud applications by using SQL Database active geo-replication

11/7/2019 • 8 minutes to read • [Edit Online](#)

Learn how to use [active geo-replication](#) in Azure SQL Database to enable rolling upgrades of your cloud application. Because upgrades are disruptive operations, they should be part of your business-continuity planning and design. In this article, we look at two different methods of orchestrating the upgrade process and discuss the benefits and tradeoffs of each option. For the purposes of this article, we refer to an application that consists of a website that's connected to a single database as its data tier. Our goal is to upgrade version 1 (V1) of the application to version 2 (V2) without any significant impact on the user experience.

When evaluating upgrade options, consider these factors:

- Impact on application availability during upgrades, such as how long application functions might be limited or degraded.
- Ability to roll back if the upgrade fails.
- Vulnerability of the application if an unrelated, catastrophic failure occurs during the upgrade.
- Total dollar cost. This factor includes additional database redundancy and incremental costs of the temporary components used by the upgrade process.

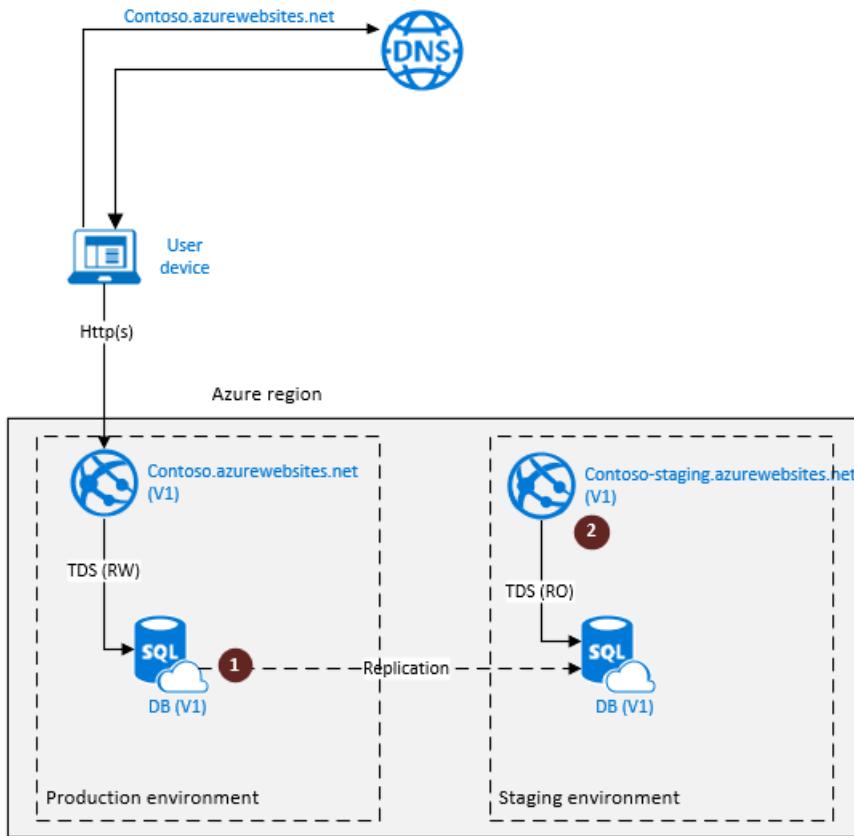
## Upgrade applications that rely on database backups for disaster recovery

If your application relies on automatic database backups and uses geo-restore for disaster recovery, it's deployed to a single Azure region. To minimize user disruption, create a staging environment in that region with all the application components involved in the upgrade. The first diagram illustrates the operational environment before the upgrade process. The endpoint `contoso.azurewebsites.net` represents a production environment of the web app. To be able to roll back the upgrade, you must create a staging environment with a fully synchronized copy of the database. Follow these steps to create a staging environment for the upgrade:

1. Create a secondary database in the same Azure region. Monitor the secondary to see if the seeding process is complete (1).
2. Create a new environment for your web app and call it 'Staging'. It will be registered in Azure DNS with the URL `contoso-staging.azurewebsites.net` (2).

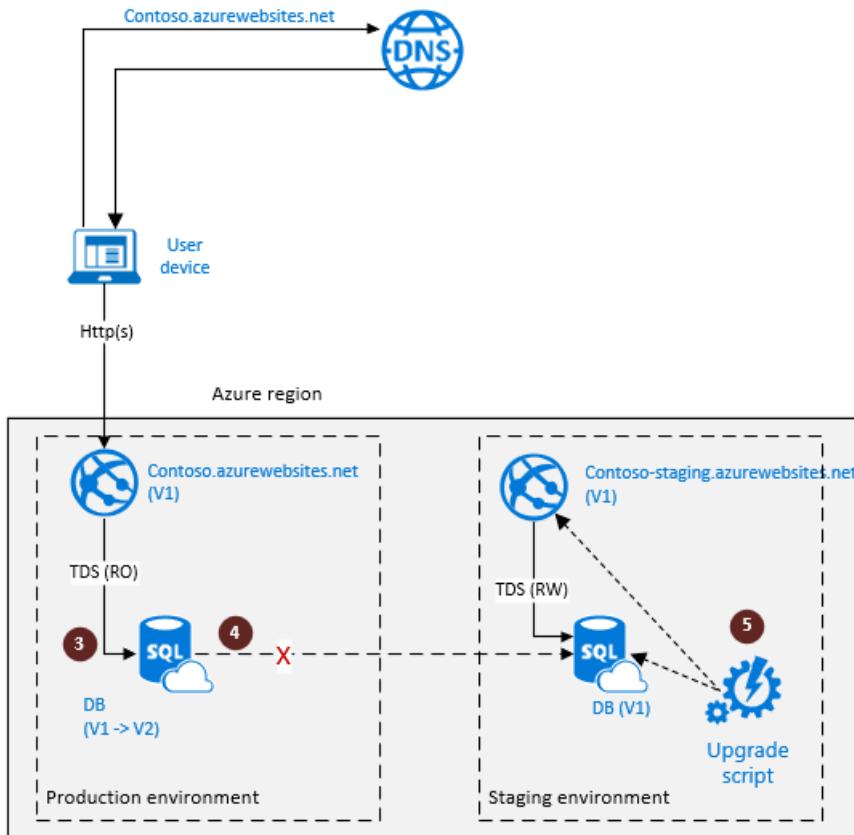
### NOTE

These preparation steps won't impact the production environment, which can function in full-access mode.



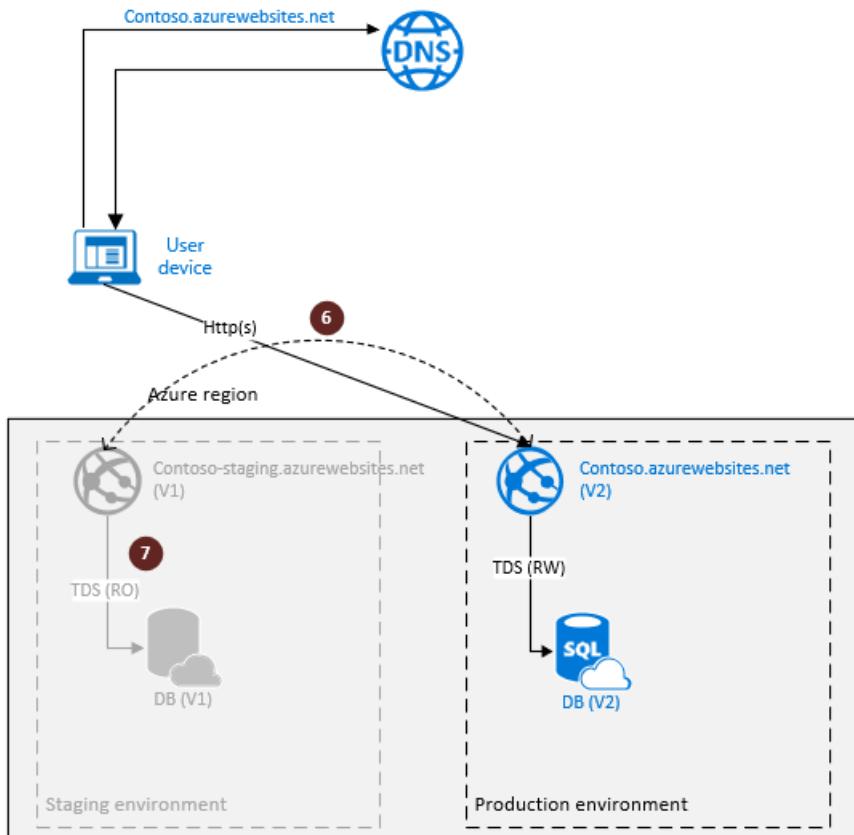
When the preparation steps are complete, the application is ready for the actual upgrade. The next diagram illustrates the steps involved in the upgrade process:

1. Set the primary database to read-only mode (3). This mode guarantees that the production environment of the web app (V1) remains read-only during the upgrade, thus preventing data divergence between the V1 and V2 database instances.
2. Disconnect the secondary database by using the planned termination mode (4). This action creates a fully synchronized, independent copy of the primary database. This database will be upgraded.
3. Turn the secondary database to read-write mode and run the upgrade script (5).



If the upgrade finishes successfully, you're now ready to switch users to the upgraded copy the application, which becomes a production environment. Switching involves a few more steps, as illustrated in the next diagram:

1. Activate a swap operation between production and staging environments of the web app (6). This operation switches the URLs of the two environments. Now `contoso.azurewebsites.net` points to the V2 version of the web site and the database (production environment).
2. If you no longer need the V1 version, which became a staging copy after the swap, you can decommission the staging environment (7).



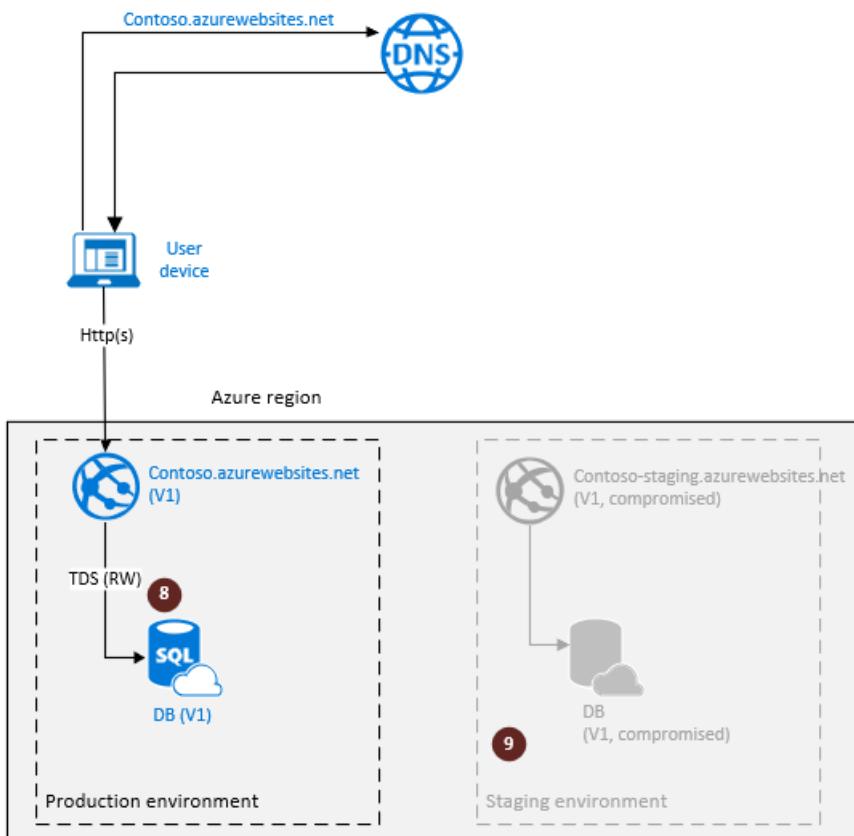
If the upgrade process is unsuccessful (for example, due to an error in the upgrade script), consider the staging environment to be compromised. To roll back the application to the pre-upgrade state, revert the application in the production environment to full access. The next diagram shows the reversion steps:

1. Set the database copy to read-write mode (8). This action restores the full V1 functionality of the production copy.
2. Perform the root-cause analysis and decommission the staging environment (9).

At this point, the application is fully functional, and you can repeat the upgrade steps.

#### **NOTE**

The rollback doesn't require DNS changes because you did not yet perform a swap operation.



The key advantage of this option is that you can upgrade an application in a single region by following a set of simple steps. The dollar cost of the upgrade is relatively low.

The main tradeoff is that, if a catastrophic failure occurs during the upgrade, the recovery to the pre-upgrade state involves redeploying the application in a different region and restoring the database from backup by using geo-restore. This process results in significant downtime.

## Upgrade applications that rely on database geo-replication for disaster recovery

If your application uses active geo-replication or auto-failover groups for business continuity, it's deployed to at least two different regions. There's an active, primary database in a primary region and a read-only, secondary database in a backup region. Along with the factors mentioned at the beginning of this article, the upgrade process must also guarantee that:

- The application remains protected from catastrophic failures at all times during the upgrade process.
- The geo-redundant components of the application are upgraded in parallel with the active components.

To achieve these goals, in addition to using the Web Apps environments, you'll take advantage of Azure Traffic Manager by using a failover profile with one active endpoint and one backup endpoint. The next diagram illustrates the operational environment prior to the upgrade process. The web sites `contoso-1.azurewebsites.net` and `contoso-dr.azurewebsites.net` represent a production environment of the application with full geographic redundancy. The production environment includes the following components:

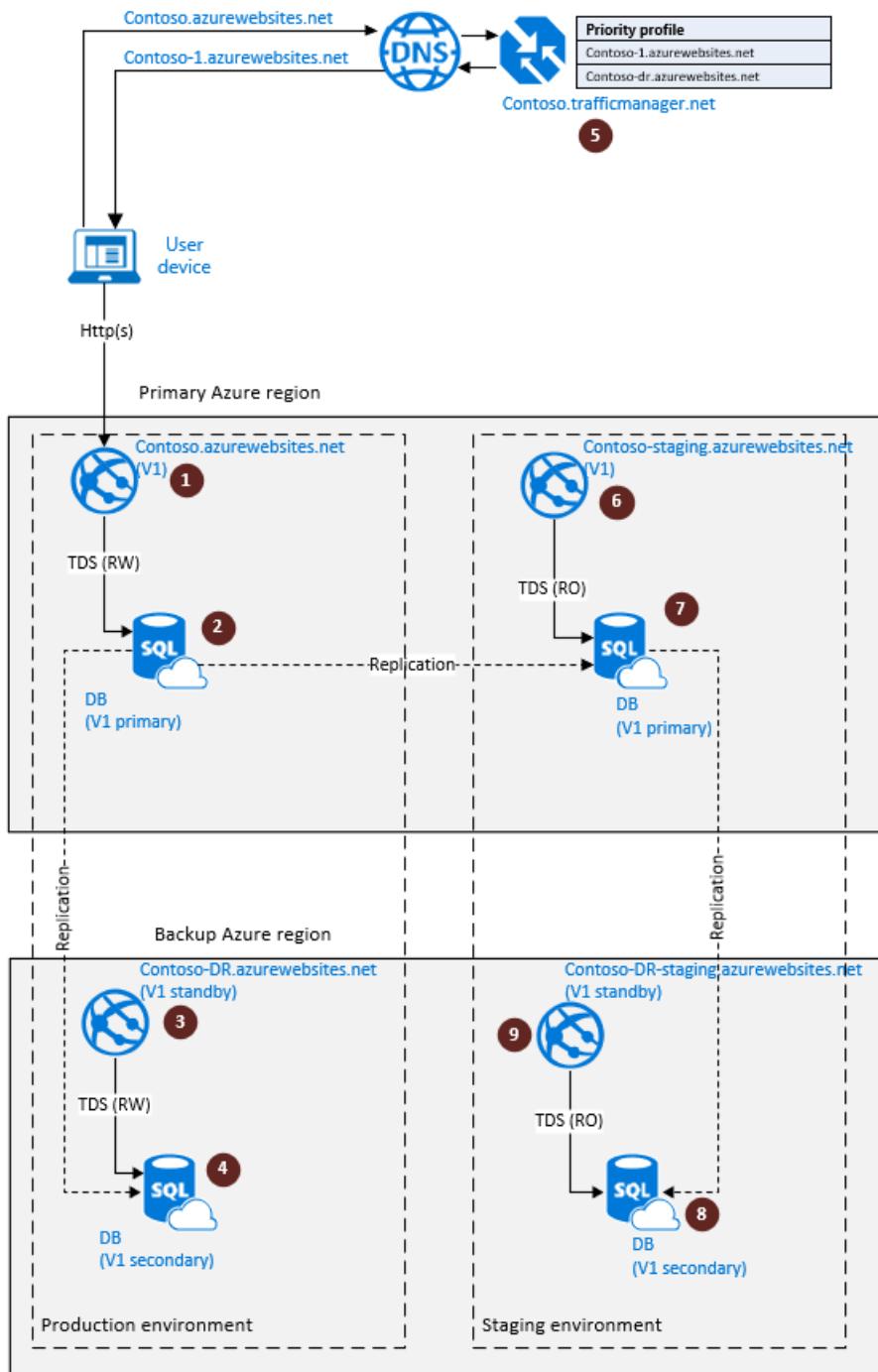
- The production environment of the web app `contoso-1.azurewebsites.net` in the primary region (1)
- The primary database in the primary region (2)
- A standby instance of the web app in the backup region (3)
- The geo-replicated secondary database in the backup region (4)
- A Traffic Manager performance profile with an online endpoint called `contoso-1.azurewebsites.net` and an offline endpoint called `contoso-dr.azurewebsites.net`

To make it possible to roll back the upgrade, you must create a staging environment with a fully synchronized copy of the application. Because you need to ensure that the application can quickly recover in case a catastrophic failure occurs during the upgrade process, the staging environment must be geo-redundant also. The following steps are required to create a staging environment for the upgrade:

1. Deploy a staging environment of the web app in the primary region (6).
2. Create a secondary database in the primary Azure region (7). Configure the staging environment of the web app to connect to it.
3. Create another geo-redundant, secondary database in the backup region by replicating the secondary database in the primary region. (This method is called *chained geo-replication*.) (8).
4. Deploy a staging environment of the web app instance in the backup region (9) and configure it to connect the geo-redundant secondary database created at (8).

**NOTE**

These preparation steps won't impact the application in the production environment. It will remain fully functional in read-write mode.



When the preparation steps are complete, the staging environment is ready for the upgrade. The next diagram illustrates these upgrade steps:

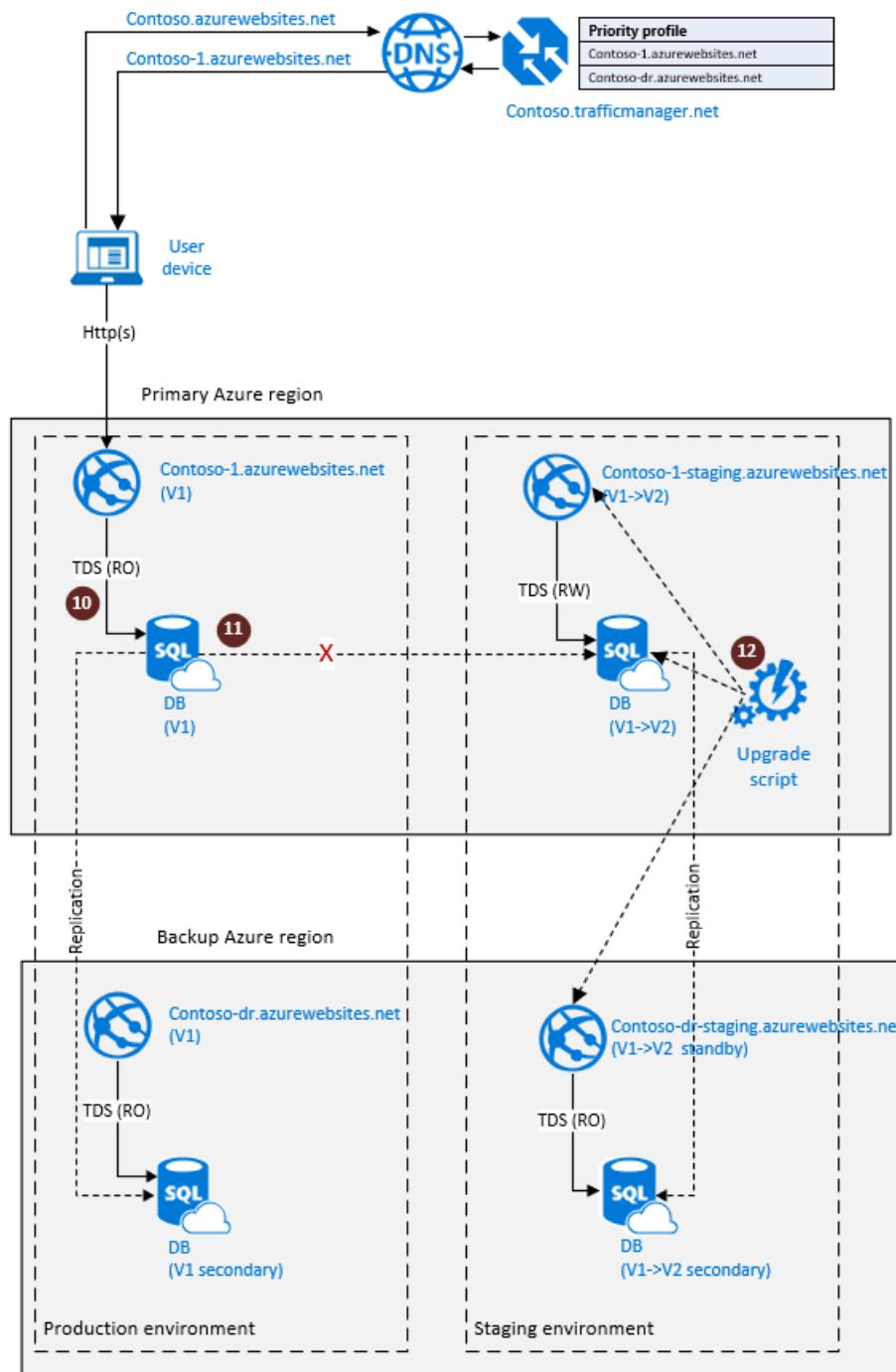
- Set the primary database in the production environment to read-only mode (10). This mode guarantees that the production database (V1) won't change during the upgrade, thus preventing the data divergence between the V1 and V2 database instances.

```
-- Set the production database to read-only mode
ALTER DATABASE <Prod_DB>
SET (ALLOW_CONNECTIONS = NO)
```

- Terminate geo-replication by disconnecting the secondary (11). This action creates an independent but fully synchronized copy of the production database. This database will be upgraded. The following example uses Transact-SQL but **PowerShell** is also available.

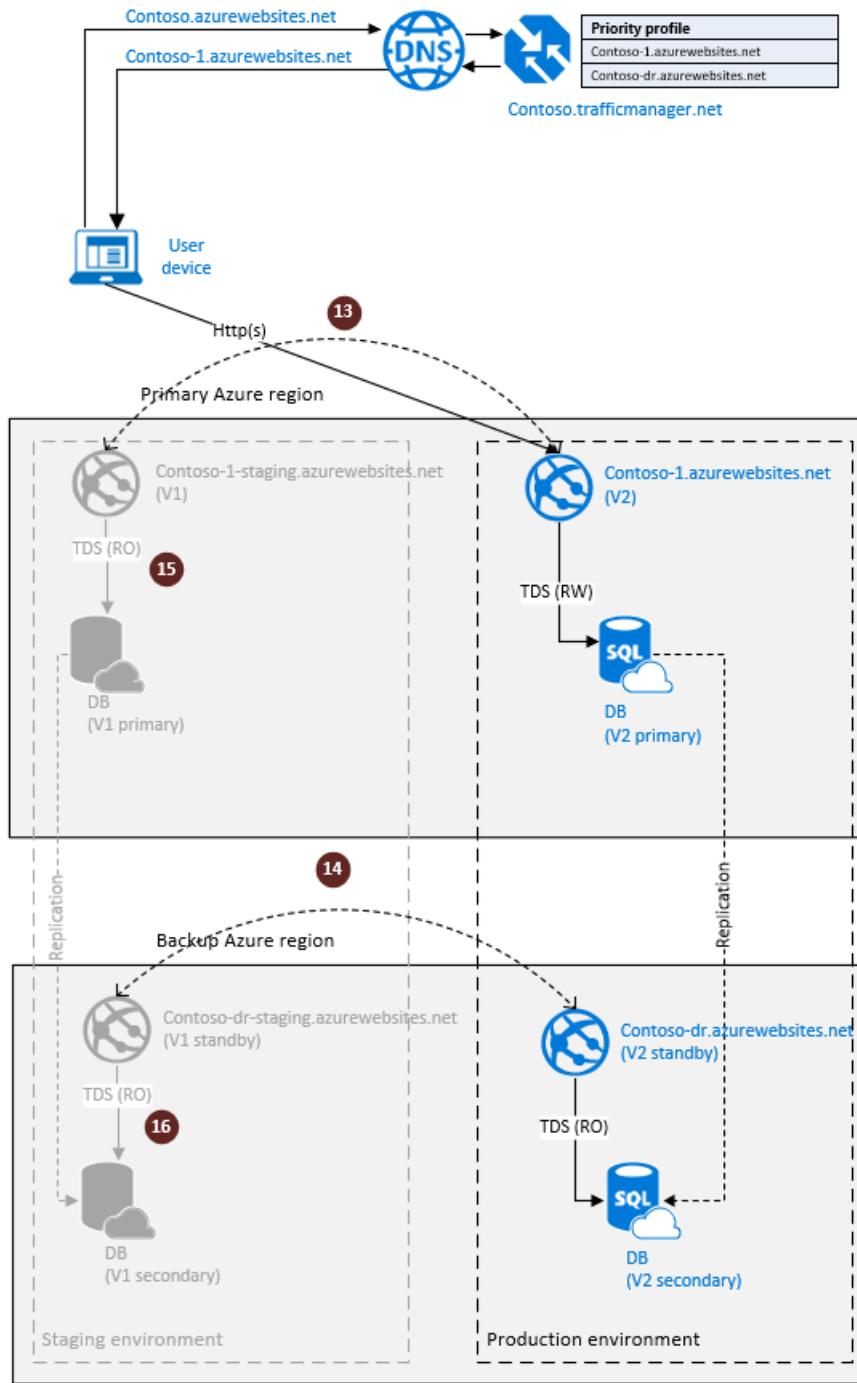
```
-- Disconnect the secondary, terminating geo-replication
ALTER DATABASE <Prod_DB>
REMOVE SECONDARY ON SERVER <Partner-Server>
```

3. Run the upgrade script against `contoso-1-staging.azurewebsites.net`, `contoso-dr-staging.azurewebsites.net`, and the staging primary database (12). The database changes will be replicated automatically to the staging secondary.



If the upgrade finishes successfully, you're now ready to switch users to the V2 version of the application. The next diagram illustrates the steps involved:

1. Activate a swap operation between production and staging environments of the web app in the primary region (13) and in the backup region (14). V2 of the application now becomes a production environment, with a redundant copy in the backup region.
2. If you no longer need the V1 application (15 and 16), you can decommission the staging environment.



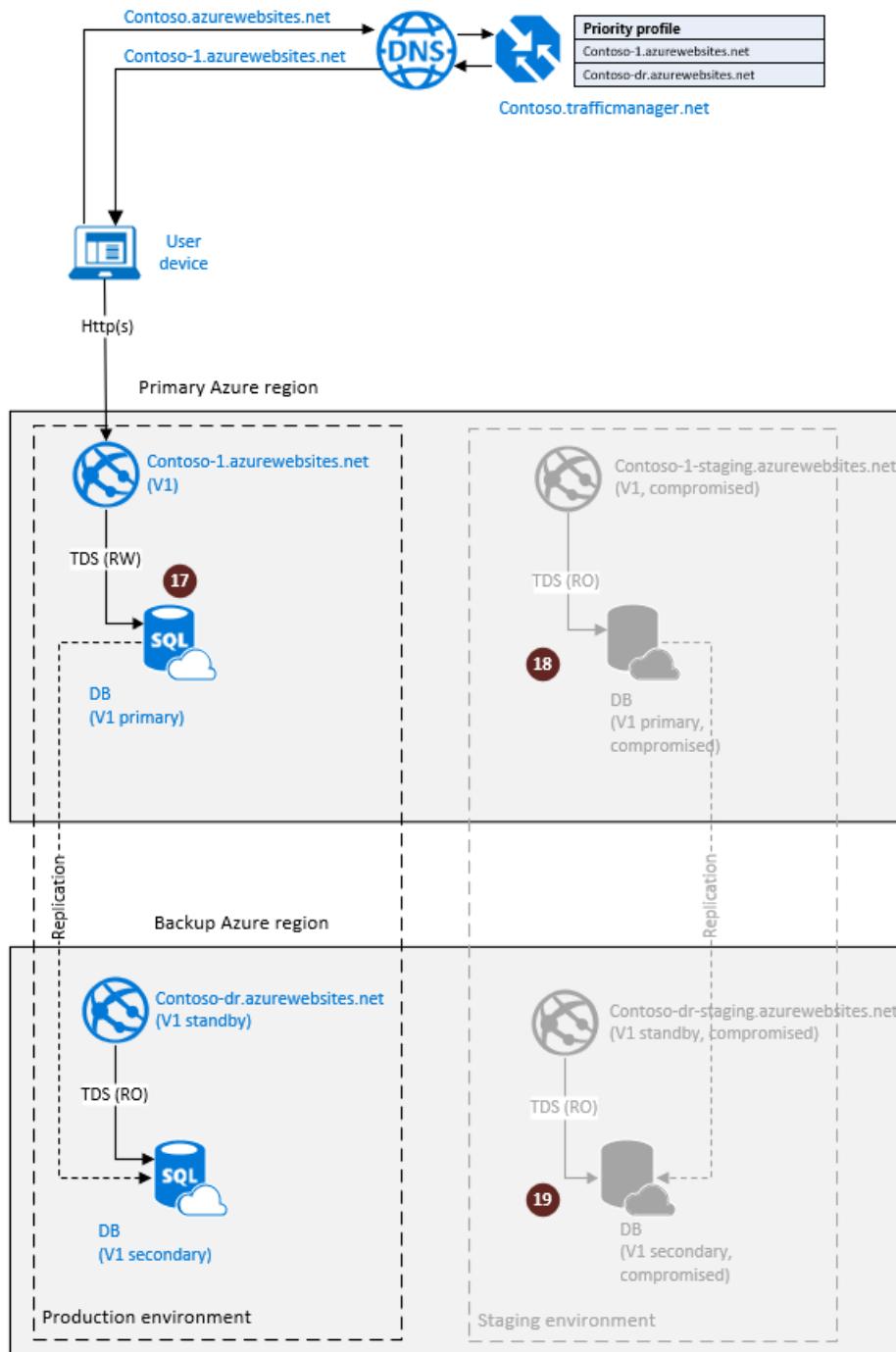
If the upgrade process is unsuccessful (for example, due to an error in the upgrade script), consider the staging environment to be in an inconsistent state. To roll back the application to the pre-upgrade state, revert to using V1 of the application in the production environment. The required steps are shown on the next diagram:

1. Set the primary database copy in the production environment to read-write mode (17). This action restores full V1 functionality in the production environment.
2. Perform the root-cause analysis and repair or remove the staging environment (18 and 19).

At this point, the application is fully functional, and you can repeat the upgrade steps.

#### NOTE

The rollback doesn't require DNS changes because you didn't perform a swap operation.



The key advantage of this option is that you can upgrade both the application and its geo-redundant copy in parallel without compromising your business continuity during the upgrade.

The main tradeoff is that it requires double redundancy of each application component and therefore incurs higher dollar cost. It also involves a more complicated workflow.

## Summary

The two upgrade methods described in the article differ in complexity and dollar cost, but they both focus on minimizing how long the user is limited to read-only operations. That time is directly defined by the duration of the upgrade script. It doesn't depend on the database size, the service tier you chose, the website configuration, or other factors that you can't easily control. All preparation steps are decoupled from the upgrade steps and don't impact the production application. The efficiency of the upgrade script is a key factor that determines the user experience during upgrades. So, the best way to improve that experience is to focus your efforts on making the upgrade script as efficient as possible.

## Next steps

- For a business continuity overview and scenarios, see [Business continuity overview](#).
- To learn about Azure SQL Database active geo-replication, see [Create readable secondary databases using active geo-replication](#).
- To learn about Azure SQL Database auto-failover groups, see [Use auto-failover groups to enable transparent and coordinated failover of multiple databases](#).
- To learn about staging environments in Azure App Service, see [Set up staging environments in Azure App Service](#).
- To learn about Azure Traffic Manager profiles, see [Manage an Azure Traffic Manager profile](#).

# Multi-tenant SaaS database tenancy patterns

11/7/2019 • 12 minutes to read • [Edit Online](#)

This article describes the various tenancy models available for a multi-tenant SaaS application.

When designing a multi-tenant SaaS application, you must carefully choose the tenancy model that best fits the needs of your application. A tenancy model determines how each tenant's data is mapped to storage. Your choice of tenancy model impacts application design and management. Switching to a different model later is sometimes costly.

## A. SaaS concepts and terminology

In the Software as a Service (SaaS) model, your company does not sell *licenses* to your software. Instead, each customer makes rent payments to your company, making each customer a *tenant* of your company.

In return for paying rent, each tenant receives access to your SaaS application components, and has its data stored in the SaaS system.

The term *tenancy model* refers to how tenants' stored data is organized:

- *Single-tenancy*: Each database stores data from only one tenant.
- *Multi-tenancy*: Each database stores data from multiple separate tenants (with mechanisms to protect data privacy).
- Hybrid tenancy models are also available.

## B. How to choose the appropriate tenancy model

In general, the tenancy model does not impact the function of an application, but it likely impacts other aspects of the overall solution. The following criteria are used to assess each of the models:

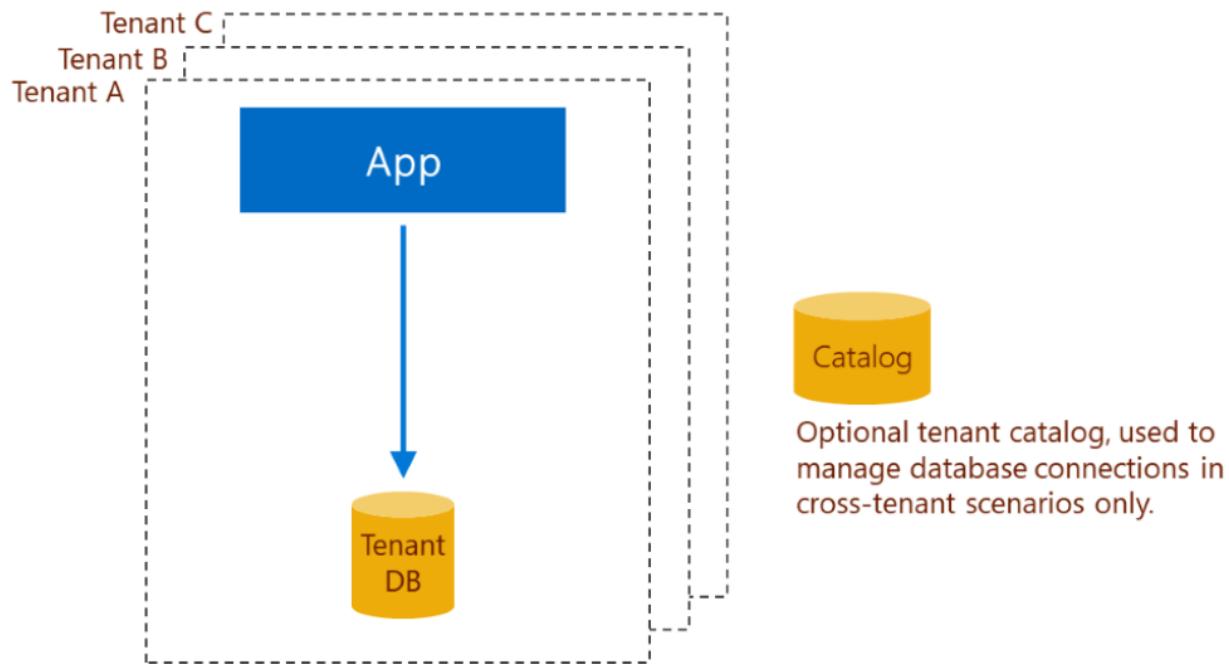
- **Scalability:**
  - Number of tenants.
  - Storage per-tenant.
  - Storage in aggregate.
  - Workload.
- **Tenant isolation:** Data isolation and performance (whether one tenant's workload impacts others).
- **Per-tenant cost:** Database costs.
- **Development complexity:**
  - Changes to schema.
  - Changes to queries (required by the pattern).
- **Operational complexity:**
  - Monitoring and managing performance.
  - Schema management.
  - Restoring a tenant.
  - Disaster recovery.
- **Customizability:** Ease of supporting schema customizations that are either tenant-specific or tenant class-specific.

The tenancy discussion is focused on the *data* layer. But consider for a moment the *application* layer. The application layer is treated as a monolithic entity. If you divide the application into many small components, your choice of tenancy model might change. You could treat some components differently than others regarding both tenancy and the storage technology or platform used.

## C. Standalone single-tenant app with single-tenant database

### Application level isolation

In this model, the whole application is installed repeatedly, once for each tenant. Each instance of the app is a standalone instance, so it never interacts with any other standalone instance. Each instance of the app has only one tenant, and therefore needs only one database. The tenant has the database all to itself.



Each app instance is installed in a separate Azure resource group. The resource group can belong to a subscription that is owned by either the software vendor or the tenant. In either case, the vendor can manage the software for the tenant. Each application instance is configured to connect to its corresponding database.

Each tenant database is deployed as a single database. This model provides the greatest database isolation. But the isolation requires that sufficient resources be allocated to each database to handle its peak loads. Here it matters that elastic pools cannot be used for databases deployed in different resource groups or to different subscriptions. This limitation makes this standalone single-tenant app model the most expensive solution from an overall database cost perspective.

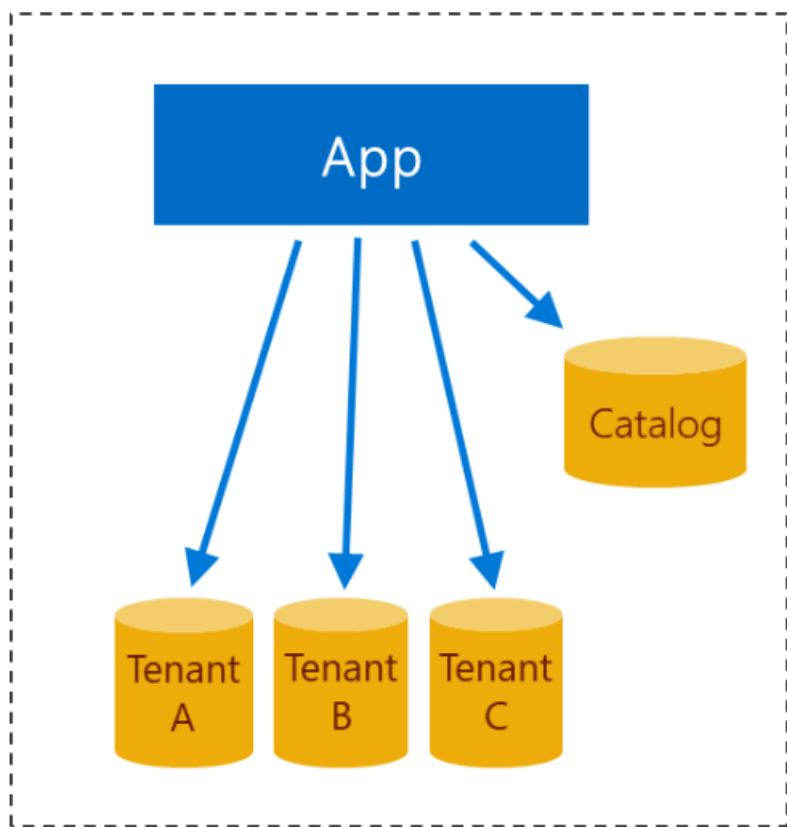
### Vendor management

The vendor can access all the databases in all the standalone app instances, even if the app instances are installed in different tenant subscriptions. The access is achieved via SQL connections. This cross-instance access can enable the vendor to centralize schema management and cross-database query for reporting or analytics purposes. If this kind of centralized management is desired, a catalog must be deployed that maps tenant identifiers to database URLs. Azure SQL Database provides a sharding library that is used together with a SQL database to provide a catalog. The sharding library is formally named the [Elastic Database Client Library](#).

## D. Multi-tenant app with database-per-tenant

This next pattern uses a multi-tenant application with many databases, all being single-tenant databases. A new database is provisioned for each new tenant. The application tier is scaled *up* vertically by adding more resources per node. Or the app is scaled *out* horizontally by adding more nodes. The scaling is based on workload, and is

independent of the number or scale of the individual databases.



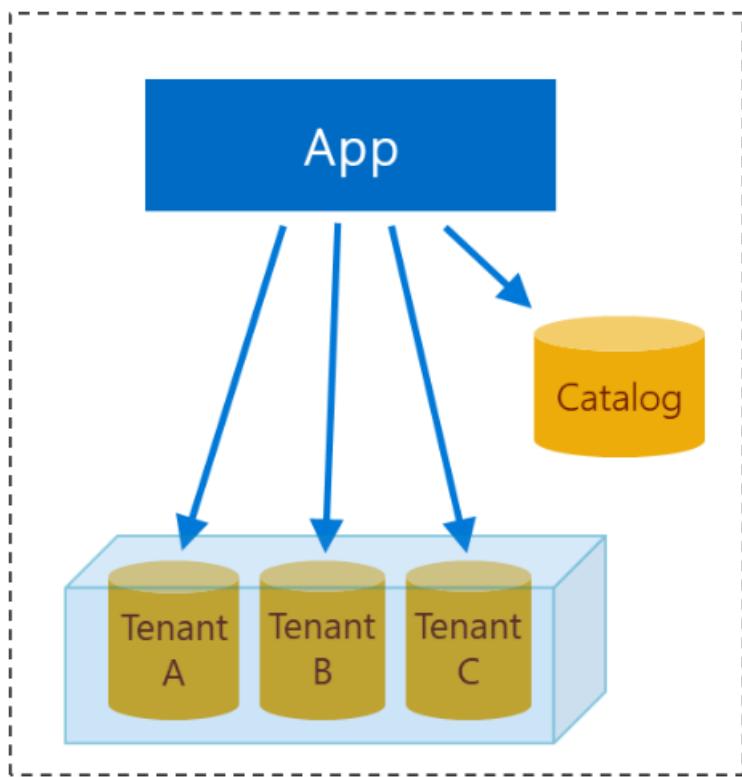
#### Customize for a tenant

Like the standalone app pattern, the use of single-tenant databases gives strong tenant isolation. In any app whose model specifies only single-tenant databases, the schema for any one given database can be customized and optimized for its tenant. This customization does not affect other tenants in the app. Perhaps a tenant might need data beyond the basic data fields that all tenants need. Further, the extra data field might need an index.

With database-per-tenant, customizing the schema for one or more individual tenants is straightforward to achieve. The application vendor must design procedures to carefully manage schema customizations at scale.

#### Elastic pools

When databases are deployed in the same resource group, they can be grouped into elastic pools. The pools provide a cost-effective way of sharing resources across many databases. This pool option is cheaper than requiring each database to be large enough to accommodate the usage peaks that it experiences. Even though pooled databases share access to resources they can still achieve a high degree of performance isolation.



Azure SQL Database provides the tools necessary to configure, monitor, and manage the sharing. Both pool-level and database-level performance metrics are available in the Azure portal, and through Azure Monitor logs. The metrics can give great insights into both aggregate and tenant-specific performance. Individual databases can be moved between pools to provide reserved resources to a specific tenant. These tools enable you to ensure good performance in a cost effective manner.

#### **Operations scale for database-per-tenant**

The Azure SQL Database platform has many management features designed to manage large numbers of databases at scale, such as well over 100,000 databases. These features make the database-per-tenant pattern plausible.

For example, suppose a system has a 1000-tenant database as its only one database. The database might have 20 indexes. If the system converts to having 1000 single-tenant databases, the quantity of indexes rises to 20,000. In SQL Database as part of [Automatic tuning](#), the automatic indexing features are enabled by default. Automatic indexing manages for you all 20,000 indexes and their ongoing create and drop optimizations. These automated actions occur within an individual database, and they are not coordinated or restricted by similar actions in other databases. Automatic indexing treats indexes differently in a busy database than in a less busy database. This type of index management customization would be impractical at the database-per-tenant scale if this huge management task had to be done manually.

Other management features that scale well include the following:

- Built-in backups.
- High availability.
- On-disk encryption.
- Performance telemetry.

#### **Automation**

The management operations can be scripted and offered through a [devops](#) model. The operations can even be automated and exposed in the application.

For example, you could automate the recovery of a single tenant to an earlier point in time. The recovery only needs to restore the one single-tenant database that stores the tenant. This restore has no impact on other tenants, which confirms that management operations are at the finely granular level of each individual tenant.

## E. Multi-tenant app with multi-tenant databases

Another available pattern is to store many tenants in a multi-tenant database. The application instance can have any number of multi-tenant databases. The schema of a multi-tenant database must have one or more tenant identifier columns so that the data from any given tenant can be selectively retrieved. Further, the schema might require a few tables or columns that are used by only a subset of tenants. However, static code and reference data is stored only once and is shared by all tenants.

### Tenant isolation is sacrificed

*Data:* A multi-tenant database necessarily sacrifices tenant isolation. The data of multiple tenants is stored together in one database. During development, ensure that queries never expose data from more than one tenant. SQL Database supports [row-level security](#), which can enforce that data returned from a query be scoped to a single tenant.

*Processing:* A multi-tenant database shares compute and storage resources across all its tenants. The database as a whole can be monitored to ensure it is performing acceptably. However, the Azure system has no built-in way to monitor or manage the use of these resources by an individual tenant. Therefore, the multi-tenant database carries an increased risk of encountering noisy neighbors, where the workload of one overactive tenant impacts the performance experience of other tenants in the same database. Additional application-level monitoring could monitor tenant-level performance.

### Lower cost

In general, multi-tenant databases have the lowest per-tenant cost. Resource costs for a single database are lower than for an equivalently sized elastic pool. In addition, for scenarios where tenants need only limited storage, potentially millions of tenants could be stored in a single database. No elastic pool can contain millions of databases. However, a solution containing 1000 databases per pool, with 1000 pools, could reach the scale of millions at the risk of becoming unwieldy to manage.

Two variations of a multi-tenant database model are discussed in what follows, with the sharded multi-tenant model being the most flexible and scalable.

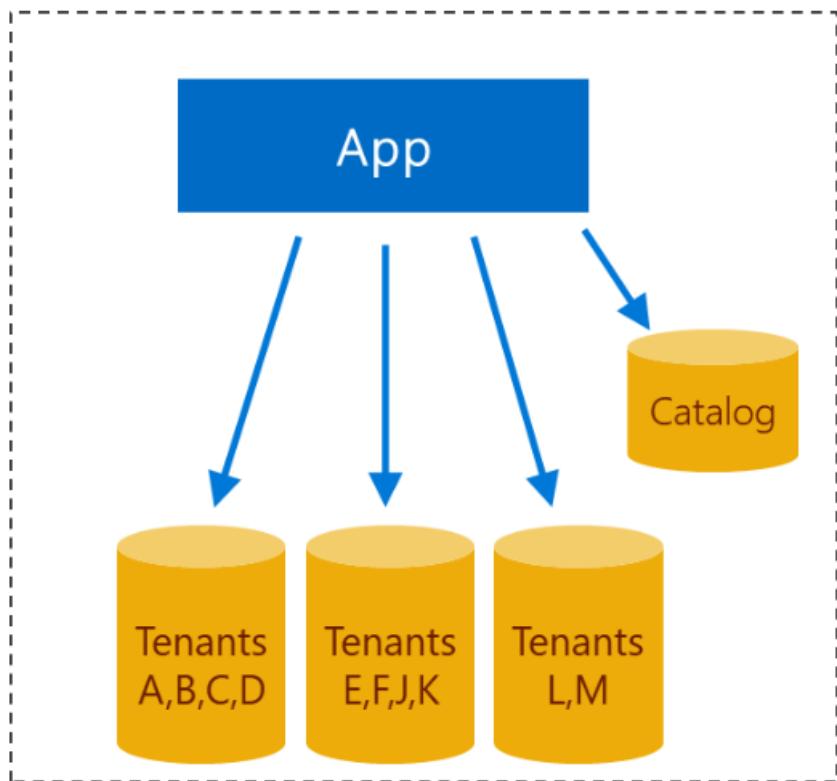
## F. Multi-tenant app with a single multi-tenant database

The simplest multi-tenant database pattern uses a single database to host data for all tenants. As more tenants are added, the database is scaled up with more storage and compute resources. This scale up might be all that is needed, although there is always an ultimate scale limit. However, long before that limit is reached the database becomes unwieldy to manage.

Management operations that are focused on individual tenants are more complex to implement in a multi-tenant database. And at scale these operations might become unacceptably slow. One example is a point-in-time restore of the data for just one tenant.

## G. Multi-tenant app with sharded multi-tenant databases

Most SaaS applications access the data of only one tenant at a time. This access pattern allows tenant data to be distributed across multiple databases or shards, where all the data for any one tenant is contained in one shard. Combined with a multi-tenant database pattern, a sharded model allows almost limitless scale.



#### **Manage shards**

Sharding adds complexity both to the design and operational management. A catalog is required in which to maintain the mapping between tenants and databases. In addition, management procedures are required to manage the shards and the tenant population. For example, procedures must be designed to add and remove shards, and to move tenant data between shards. One way to scale is to by adding a new shard and populating it with new tenants. At other times you might split a densely populated shard into two less-densely populated shards. After several tenants have been moved or discontinued, you might merge sparsely populated shards together. The merge would result in more cost-efficient resource utilization. Tenants might also be moved between shards to balance workloads.

SQL Database provides a split/merge tool that works in conjunction with the sharding library and the catalog database. The provided app can split and merge shards, and it can move tenant data between shards. The app also maintains the catalog during these operations, marking affected tenants as offline prior to moving them. After the move, the app updates the catalog again with the new mapping, and marking the tenant as back online.

#### **Smaller databases more easily managed**

By distributing tenants across multiple databases, the sharded multi-tenant solution results in smaller databases that are more easily managed. For example, restoring a specific tenant to a prior point in time now involves restoring a single smaller database from a backup, rather than a larger database that contains all tenants. The database size, and number of tenants per database, can be chosen to balance the workload and the management efforts.

#### **Tenant identifier in the schema**

Depending on the sharding approach used, additional constraints may be imposed on the database schema. The SQL Database split/merge application requires that the schema includes the sharding key, which typically is the tenant identifier. The tenant identifier is the leading element in the primary key of all sharded tables. The tenant identifier enables the split/merge application to quickly locate and move data associated with a specific tenant.

#### **Elastic pool for shards**

Sharded multi-tenant databases can be placed in elastic pools. In general, having many single-tenant databases in a pool is as cost efficient as having many tenants in a few multi-tenant databases. Multi-tenant databases are advantageous when there are a large number of relatively inactive tenants.

## H. Hybrid sharded multi-tenant database model

In the hybrid model, all databases have the tenant identifier in their schema. The databases are all capable of storing more than one tenant, and the databases can be sharded. So in the schema sense, they are all multi-tenant databases. Yet in practice some of these databases contain only one tenant. Regardless, the quantity of tenants stored in a given database has no effect on the database schema.

### Move tenants around

At any time, you can move a particular tenant to its own multi-tenant database. And at any time, you can change your mind and move the tenant back to a database that contains multiple tenants. You can also assign a tenant to new single-tenant database when you provision the new database.

The hybrid model shines when there are large differences between the resource needs of identifiable groups of tenants. For example, suppose that tenants participating in a free trial are not guaranteed the same high level of performance that subscribing tenants are. The policy might be for tenants in the free trial phase to be stored in a multi-tenant database that is shared among all the free trial tenants. When a free trial tenant subscribes to the basic service tier, the tenant can be moved to another multi-tenant database that might have fewer tenants. A subscriber that pays for the premium service tier could be moved to its own new single-tenant database.

### Pools

In this hybrid model, the single-tenant databases for subscriber tenants can be placed in resource pools to reduce database costs per tenant. This is also done in the database-per-tenant model.

## I. Tenancy models compared

The following table summarizes the differences between the main tenancy models.

MEASUREMENT	STANDALONE APP	DATABASE-PER-TENANT	SHARDED MULTI-TENANT
Scale	Medium 1-100s	Very high 1-100,000s	Unlimited 1-1,000,000s
Tenant isolation	Very high	High	Low; except for any single tenant (that is alone in an MT db).
Database cost per tenant	High; is sized for peaks.	Low; pools used.	Lowest, for small tenants in MT DBs.
Performance monitoring and management	Per-tenant only	Aggregate + per-tenant	Aggregate; although is per-tenant only for singles.
Development complexity	Low	Low	Medium; due to sharding.
Operational complexity	Low-High. Individually simple, complex at scale.	Low-Medium. Patterns address complexity at scale.	Low-High. Individual tenant management is complex.

## Next steps

- [Deploy and explore a multi-tenant Wingtip application that uses the database-per-tenant SaaS model - Azure SQL Database](#)
- [Welcome to the Wingtip Tickets sample SaaS Azure SQL Database tenancy app](#)

# Video indexed and annotated for multi-tenant SaaS app using Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

This article is an annotated index into the time locations of an 81 minute video about SaaS tenancy models or patterns. This article enables you to skip backward or forward in the video to which portion interests you. The video explains the major design options for a multi-tenant database application on Azure SQL Database. The video includes demos, walkthroughs of management code, and at times more detail informed by experience than might be in our written documentation.

The video amplifies information in our written documentation, found at:

- *Conceptual:* [Multi-tenant SaaS database tenancy patterns](#)
- *Tutorials:* [The Wingtip Tickets SaaS application](#)

The video and the articles describe the many phases of creating a multi-tenant application on Azure SQL Database in the cloud. Special features of Azure SQL Database make it easier to develop and implement multi-tenant apps that are both easier to manage and reliably performant.

We routinely update our written documentation. The video is not edited or updated, so eventually more of its detail may become outdated.

## Sequence of 38 time-indexed screenshots

This section indexes the time location for 38 discussions throughout the 81 minute video. Each time index is annotated with a screenshot from the video, and sometimes with additional information.

Each time index is in the format of *h:mm:ss*. For instance, the second indexed time location, labeled **Session objectives**, starts at the approximate time location of **0:03:11**.

### Compact links to video indexed time locations

The following titles are links to their corresponding annotated sections later in this article:

- 1. ([Start](#)) Welcome slide, 0:00:03
- 2. [Session objectives](#), 0:03:11
- 3. [Agenda](#), 0:04:17
- 4. [Multi-tenant web app](#), 0:05:05
- 5. [App web form in action](#), 0:05:55
- 6. [Per-tenant cost \(scale, isolation, recovery\)](#), 0:09:31
- 7. [Database models for multi-tenant: pros and cons](#), 0:11:59
- 8. [Hybrid model blends benefits of MT/ST](#), 0:13:01
- 9. [Single-tenant vs multi-tenant: pros and cons](#), 0:16:44
- 10. [Pools are cost-effective for unpredictable workloads](#), 0:19:36
- 11. [Demo of database-per-tenant and hybrid ST/MT](#), 0:20:08
- 12. [Live app form showing Dojo](#), 0:20:29
- 13. [MYOB and not a DBA in sight](#), 0:28:54
- 14. [MYOB elastic pool usage example](#), 0:29:40
- 15. [Learning from MYOB and other ISVs](#), 0:31:36
- 16. [Patterns compose into E2E SaaS scenario](#), 0:43:15

- 17. Canonical hybrid multi-tenant SaaS app, 0:47:33
- 18. Wingtip SaaS sample app, 0:48:10
- 19. Scenarios and patterns explored in the tutorials, 0:49:10
- 20. Demo of tutorials and GitHub repository, 0:50:18
- 21. GitHub repo Microsoft/WingtipSaaS, 0:50:38
- 22. Exploring the patterns, 0:56:20
- 23. Provisioning tenants and onboarding, 0:57:44
- 24. Provisioning tenants and application connection, 0:58:58
- 25. Demo of management scripts provisioning a single tenant, 0:59:43
- 26. PowerShell to provision and catalog, 1:00:02
- 27. T-SQL SELECT \* FROM TenantsExtended, 1:03:30
- 28. Managing unpredictable tenant workloads, 1:04:36
- 29. Elastic pool monitoring, 1:06:39
- 30. Load generation and performance monitoring, 1:09:42
- 31. Schema management at scale, 1:10:33
- 32. Distributed query across tenant databases, 1:12:21
- 33. Demo of ticket generation, 1:12:32
- 34. SSMS adhoc analytics, 1:12:46
- 35. Extract tenant data into SQL DW, 1:16:32
- 36. Graph of daily sale distribution, 1:16:48
- 37. Wrap up and call to action, 1:19:52
- 38. Resources for more information, 1:20:42

#### **Annotated index time locations in the video**

Clicking any screenshot image takes you to the exact time location in the video.

##### **1. (Start) Welcome slide, 0:00:01**

*Learning from MYOB: Design patterns for SaaS applications on Azure SQL Database - BRK3120*



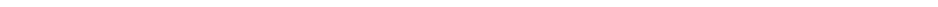
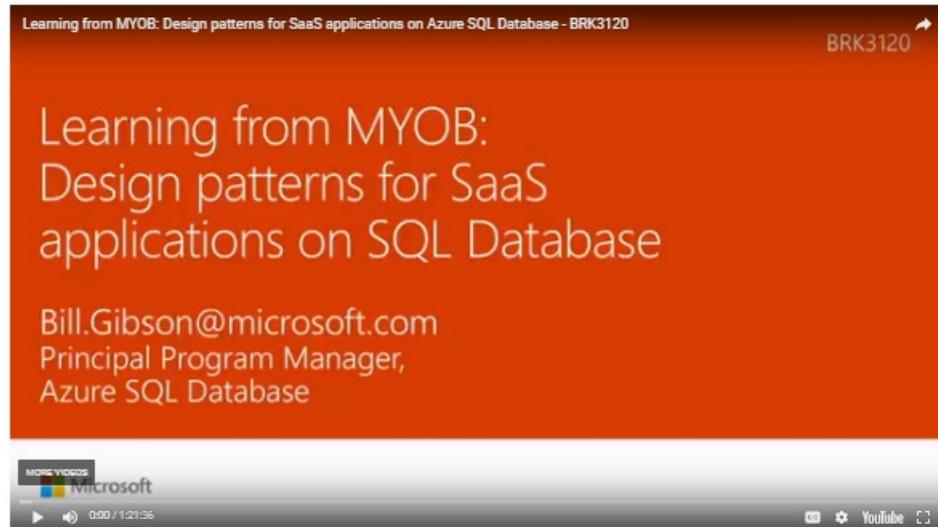
Channel 9 | all content shows events search channel 9 sign in

Microsoft Ignite 2017

Learning from MYOB: Design patterns for SaaS applications on Azure SQL Database

Oct 11, 2017 at 12:45PM by Bill Gibson

★ ★ ★ ★ 0 ratings



Description Share

Many customers are developing SaaS apps on Azure SQL Database. In working with customers like MYOB, an accounting ISV, Microsoft has identified a series of SaaS patterns that accelerate SaaS

- Title: Learning from MYOB: Design patterns for SaaS applications on Azure SQL Database
- Bill.Gibson@microsoft.com
- Principal Program Manager, Azure SQL Database
- Microsoft Ignite session BRK3120, Orlando, FL USA, October/11 2017

## 2. Session objectives, 0:01:53



Session objectives and takeaway

Equip you with...

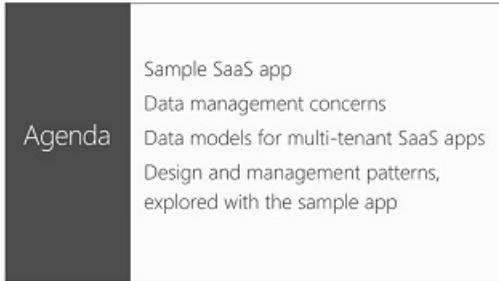
- Alternative database models for multi-tenant apps, with pros and cons
- SaaS patterns to reduce development, management and resource costs
- A sample app + scripts

Takeaway

- PaaS features + SaaS patterns make SQL DB a highly scalable, cost-efficient data platform for multi-tenant SaaS

- Alternative models for multi-tenant apps, with pros and cons.
- SaaS patterns to reduce development, management, and resource costs.
- A sample app + scripts.
- PaaS features + SaaS patterns make SQL Database a highly scalable, cost-efficient data platform for multi-tenant SaaS.

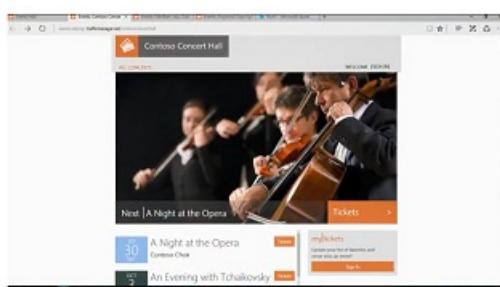
## 3. Agenda, 0:04:09



#### 4. Multi-tenant web app, 0:05:00



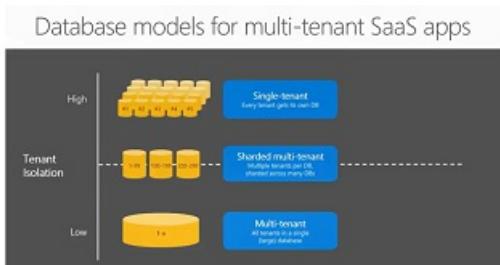
#### 5. App web form in action, 0:05:39



#### 6. Per-tenant cost (scale, isolation, recovery), 0:06:58

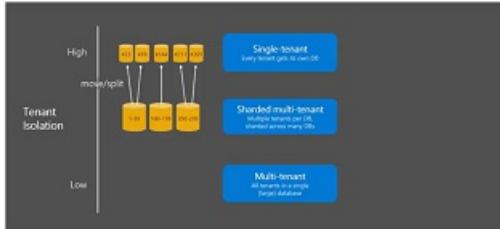


#### 7. Database models for multi-tenant: pros and cons, 0:09:52



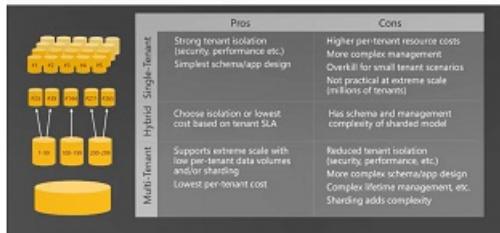
#### 8. Hybrid model blends benefits of MT/ST, 0:12:29

## Hybrid model blends benefits of MT/ST



## 9. Single-tenant vs multi-tenant: pros and cons, 0:13:11

### Single-tenant vs multi-tenant: pros and cons



## 10. Pools are cost-effective for unpredictable workloads, 0:17:49

### Pools are cost-effective for unpredictable workloads



## 11. Demo of database-per-tenant and hybrid ST/MT, 0:19:59

Demo

Wingtip SaaS sample app

- Database-per-tenant version
- Hybrid multi-tenant version

## 12. Live app form showing Dojo, 0:20:10



### 13. MYOB and not a DBA in sight, 0:25:06

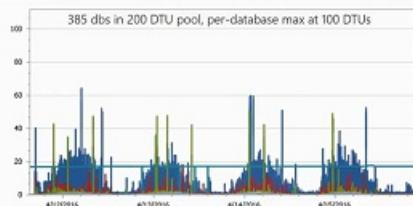
#### MYOB

- Australian accounting ISV, targeting small/medium businesses
- Migrated an established desktop app to the cloud
- Client/server app migrated easily to a single-tenant model
- Initially used standalone databases, now using databases in elastic pools
- Considered a multi-tenant database solution (did a POC)
- Pool-based solution had lower TCO and let them focus on their business
- 150,000 tenant databases in 300+ elastic pools
- Growing at 3-5K databases per month

....and not a DBA in sight !

### 14. MYOB elastic pool usage example, 0:29:30

#### MYOB elastic pool usage example

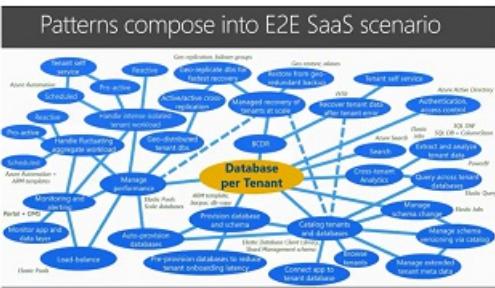


### 15. Learning from MYOB and other ISVs, 0:31:25

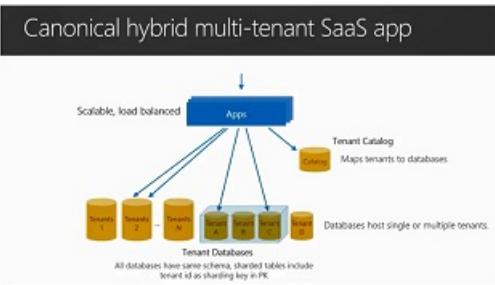
Learning from MYOB and other ISVs

SaaS patterns that address key design  
and management scenarios that are  
important at scale

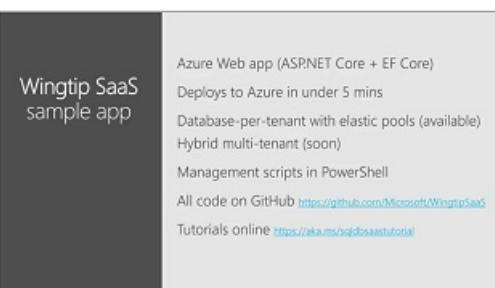
### 16. Patterns compose into E2E SaaS scenario, 0:31:42



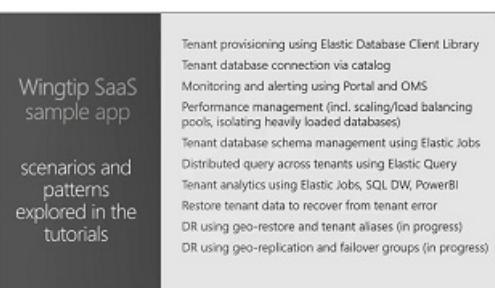
## 17. Canonical hybrid multi-tenant SaaS app, 0:46:04



## 18. Wingtip SaaS sample app, 0:48:01



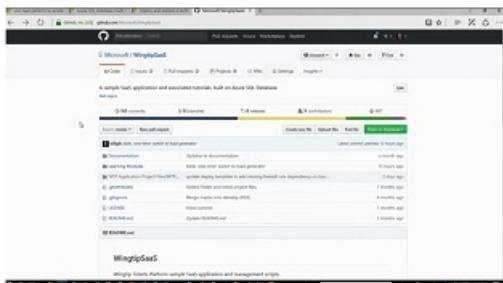
## 19. Scenarios and patterns explored in the tutorials, 0:49:00



## 20. Demo of tutorials and GitHub repository, 0:50:12



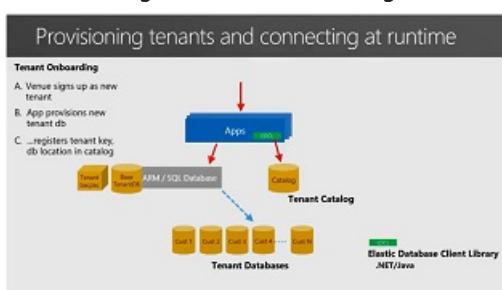
## 21. GitHub repo Microsoft/WingtipSaaS, 0:50:32



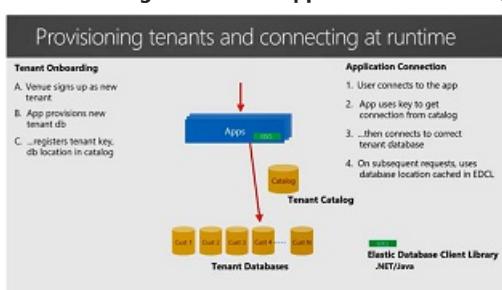
## 22. Exploring the patterns, 0:56:15



## 23. Provisioning tenants and onboarding, 0:56:19



## 24. Provisioning tenants and application connection, 0:57:52

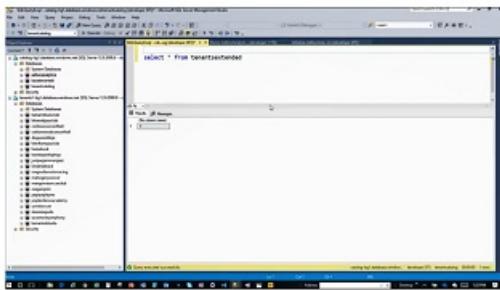


## 25. Demo of management scripts provisioning a single tenant, 0:59:36

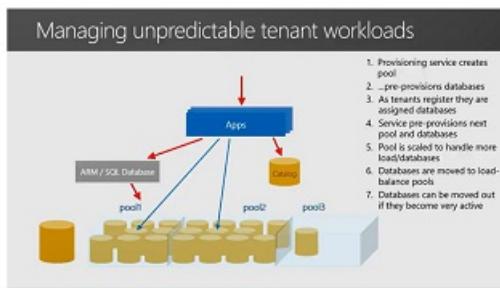


26. PowerShell to provision and catalog, 0:59:56

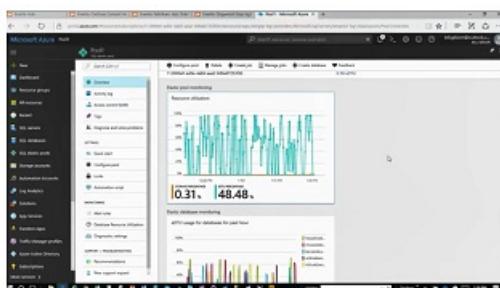
27. T-SQL SELECT \* FROM TenantsExtended, 1:03:25



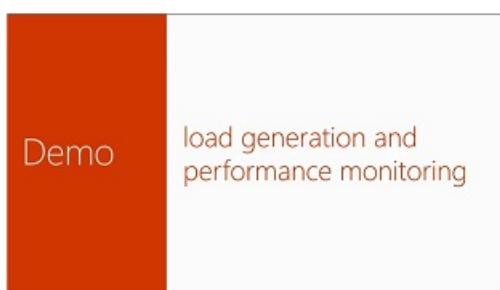
28. Managing unpredictable tenant workloads, 1:03:34



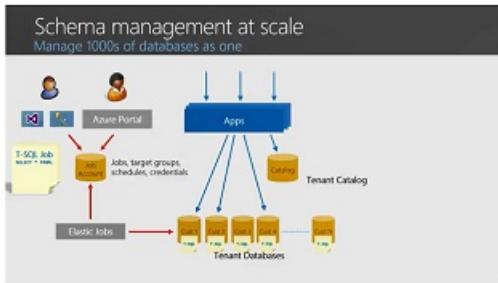
29. Elastic pool monitoring, 1:06:32



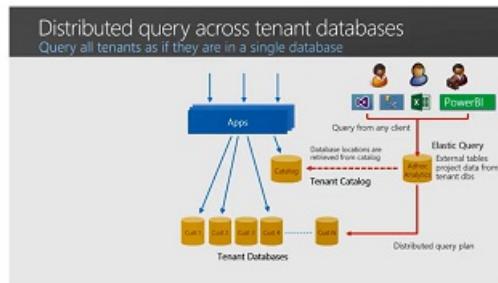
30. Load generation and performance monitoring, 1:09:37



### 31. Schema management at scale, 1:09:40



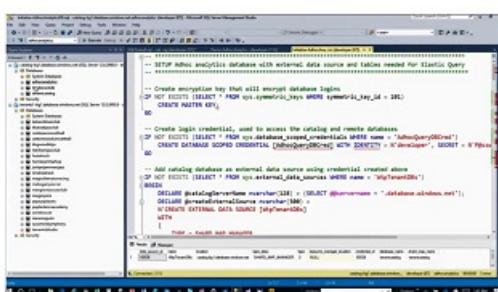
### 32. Distributed query across tenant databases, 1:11:18



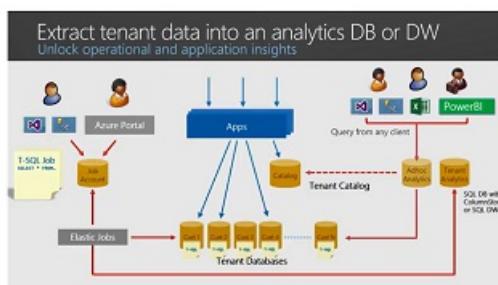
### 33. Demo of ticket generation, 1:12:28



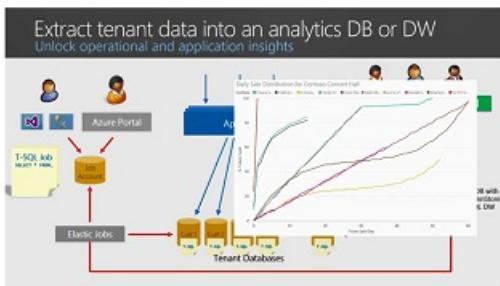
### 34. SSMS adhoc analytics, 1:12:35



### 35. Extract tenant data into SQL DW, 1:15:46



### 36. Graph of daily sale distribution, 1:16:38



### 37. Wrap up and call to action, 1:17:43

**Wrap up and call to action**

You should now be equipped...

- To choose the SaaS data model appropriate to your scenario
- To explore SaaS-specific patterns that can reduce development, management and resource costs, and time-to-market

**Takeaway**

- PaaS features + SaaS patterns make SQL DB a highly scalable, cost-efficient data platform for multi-tenant SaaS

Install the sample, try the tutorials, send us feedback  
saasfeedback@microsoft.com

### 38. Resources for more information, 1:20:35

**Resources**

- Blog post <https://azure.microsoft.com/en-us/blog/saas-patterns-accelerate-saas-application-development-on-sql-database/>
- GitHub repo <https://github.com/microsoft/wingtipsaas>
- Tutorials <https://aka.ms/goldsaastutorial>
- Getting started guide - installing and exploring the app <https://docs.microsoft.com/en-us/azure/sql-database/sql-database-saas-tutorial>

- [Blog post, May 22, 2017](#)
- [Conceptual: Multi-tenant SaaS database tenancy patterns](#)
- [Tutorials: The Wingtip Tickets SaaS application](#)
- GitHub repositories for flavors of the Wingtip Tickets SaaS tenancy application:
  - [GitHub repo for - Standalone application model.](#)
  - [GitHub repo for - DB Per Tenant model.](#)
  - [GitHub repo for - Multi-Tenant DB model.](#)

## Next steps

- [First tutorial article](#)

# Multi-tenant applications with elastic database tools and row-level security

11/7/2019 • 11 minutes to read • [Edit Online](#)

Elastic database tools and row-level security (RLS) cooperate to enable scaling the data tier of a multi-tenant application with Azure SQL Database. Together these technologies help you build an application that has a highly scalable data tier. The data tier supports multi-tenant shards, and uses **ADO.NET SqlClient** or **Entity Framework**.

For more information, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).

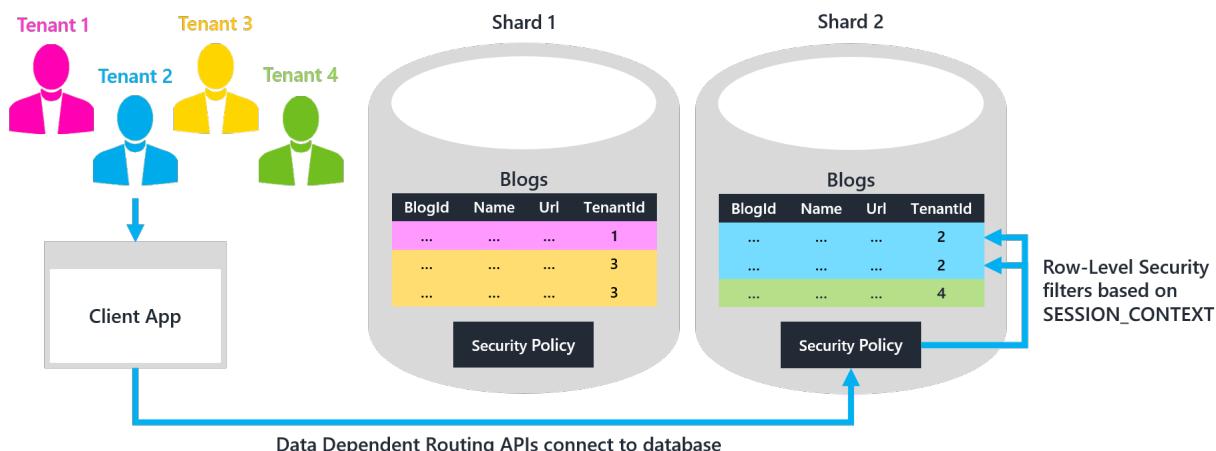
- **Elastic database tools** enable developers to scale out the data tier with standard sharding practices, by using .NET libraries and Azure service templates. Managing shards by using the [Elastic Database Client Library](#) helps automate and streamline many of the infrastructural tasks typically associated with sharding.
- **Row-level security** enables developers to safely store data for multiple tenants in the same database. RLS security policies filter out rows that do not belong to the tenant executing a query. Centralizing the filter logic inside the database simplifies maintenance and reduces the risk of a security error. The alternative of relying on all client code to enforce security is risky.

By using these features together, an application can store data for multiple tenants in the same shard database. It costs less per tenant when the tenants share a database. Yet the same application can also offer its premium tenants the option of paying for their own dedicated single-tenant shard. One benefit of single-tenant isolation is firmer performance guarantees. In a single-tenant database, there is no other tenant competing for resources.

The goal is to use the elastic database client library [data-dependent routing](#) APIs to automatically connect each given tenant to the correct shard database. Only one shard contains particular TenantId value for the given tenant. The TenantId is the *sharding key*. After the connection is established, an RLS security policy within the database ensures that the given tenant can access only those data rows that contain its TenantId.

## NOTE

The tenant identifier might consist of more than one column. For convenience in this discussion, we informally assume a single-column TenantId.



[Download the sample project](#)

[Prerequisites](#)

- Use Visual Studio (2012 or higher)
- Create three Azure SQL databases
- Download sample project: [Elastic DB Tools for Azure SQL - Multi-Tenant Shards](#)
  - Fill in the information for your databases at the beginning of **Program.cs**

This project extends the one described in [Elastic DB Tools for Azure SQL - Entity Framework Integration](#) by adding support for multi-tenant shard databases. The project builds a simple console application for creating blogs and posts. The project includes four tenants, plus two multi-tenant shard databases. This configuration is illustrated in the preceding diagram.

Build and run the application. This run bootstraps the elastic database tools' shard map manager, and performs the following tests:

1. Using Entity Framework and LINQ, create a new blog and then display all blogs for each tenant
2. Using ADO.NET SqlClient, display all blogs for a tenant
3. Try to insert a blog for the wrong tenant to verify that an error is thrown

Notice that because RLS has not yet been enabled in the shard databases, each of these tests reveals a problem: tenants are able to see blogs that do not belong to them, and the application is not prevented from inserting a blog for the wrong tenant. The remainder of this article describes how to resolve these problems by enforcing tenant isolation with RLS. There are two steps:

1. **Application tier:** Modify the application code to always set the current TenantId in the SESSION\_CONTEXT after opening a connection. The sample project already sets the TenantId this way.
2. **Data tier:** Create an RLS security policy in each shard database to filter rows based on the TenantId stored in SESSION\_CONTEXT. Create a policy for each of your shard databases, otherwise rows in multi-tenant shards are not be filtered.

## 1. Application tier: Set TenantId in the SESSION\_CONTEXT

First you connect to a shard database by using the data-dependent routing APIs of the elastic database client library. The application still must tell the database which TenantId is using the connection. The TenantId tells the RLS security policy which rows must be filtered out as belonging to other tenants. Store the current TenantId in the SESSION\_CONTEXT of the connection.

An alternative to SESSION\_CONTEXT is to use [CONTEXT\\_INFO](#). But SESSION\_CONTEXT is a better option. SESSION\_CONTEXT is easier to use, it returns NULL by default, and it supports key-value pairs.

### Entity Framework

For applications using Entity Framework, the easiest approach is to set the SESSION\_CONTEXT within the ElasticScaleContext override described in [Data-dependent routing using EF DbContext](#). Create and execute a SqlCommand that sets TenantId in the SESSION\_CONTEXT to the shardingKey specified for the connection. Then return the connection brokered through data-dependent routing. This way, you only need to write code once to set the SESSION\_CONTEXT.

```

// ElasticScaleContext.cs
// Constructor for data-dependent routing.
// This call opens a validated connection that is routed to the
// proper shard by the shard map manager.
// Note that the base class constructor call fails for an open connection
// if migrations need to be done and SQL credentials are used.
// This is the reason for the separation of constructors.
// ...
public ElasticScaleContext(ShardMap shardMap, T shardingKey, string connectionStr)
 : base(
 OpenDDRConnection(shardMap, shardingKey, connectionStr),
 true) // contextOwnsConnection
{
}

public static SqlConnection OpenDDRConnection(
 ShardMap shardMap,
 T shardingKey,
 string connectionStr)
{
 // No initialization.
 Database.SetInitializer<ElasticScaleContext<T>>(null);

 // Ask shard map to broker a validated connection for the given key.
 SqlConnection conn = null;
 try
 {
 conn = shardMap.OpenConnectionForKey(
 shardingKey,
 connectionStr,
 ConnectionOptions.Validate);

 // Set TenantId in SESSION_CONTEXT to shardingKey
 // to enable Row-Level Security filtering.
 SqlCommand cmd = conn.CreateCommand();
 cmd.CommandText =
 @"exec sp_set_session_context
 @key=N'TenantId', @value=@shardingKey";
 cmd.Parameters.AddWithValue("@shardingKey", shardingKey);
 cmd.ExecuteNonQuery();

 return conn;
 }
 catch (Exception)
 {
 if (conn != null)
 {
 conn.Dispose();
 }
 throw;
 }
}
// ...

```

Now the SESSION\_CONTEXT is automatically set with the specified TenantId whenever ElasticScaleContext is invoked:

```
// Program.cs
SqlDatabaseUtils.SqlRetryPolicy.ExecuteAction(() =>
{
 using (var db = new ElasticScaleContext<int>(
 sharding.ShardMap, tenantId, connStrBldr.ConnectionString))
 {
 var query = from b in db.Blogs
 orderby b.Name
 select b;

 Console.WriteLine("All blogs for TenantId {0}:", tenantId);
 foreach (var item in query)
 {
 Console.WriteLine(item.Name);
 }
 }
});
```

## ADO.NET SqlClient

For applications using ADO.NET SqlConnection, create a wrapper function around method `ShardMap.OpenConnectionForKey`. Have the wrapper automatically set TenantId in the SESSION\_CONTEXT to the current TenantId before returning a connection. To ensure that SESSION\_CONTEXT is always set, you should only open connections using this wrapper function.

```

// Program.cs
// Wrapper function for ShardMap.OpenConnectionForKey() that
// automatically sets SESSION_CONTEXT with the correct
// tenantId before returning a connection.
// As a best practice, you should only open connections using this method
// to ensure that SESSION_CONTEXT is always set before executing a query.
// ...
public static SqlConnection OpenConnectionForTenant(
 ShardMap shardMap, int tenantId, string connectionStr)
{
 SqlConnection conn = null;
 try
 {
 // Ask shard map to broker a validated connection for the given key.
 conn = shardMap.OpenConnectionForKey(
 tenantId, connectionStr, ConnectionOptions.Validate);

 // Set TenantId in SESSION_CONTEXT to shardingKey
 // to enable Row-Level Security filtering.
 SqlCommand cmd = conn.CreateCommand();
 cmd.CommandText =
 @"exec sp_set_session_context
 @key=N'TenantId', @value=@shardingKey";
 cmd.Parameters.AddWithValue("@shardingKey", tenantId);
 cmd.ExecuteNonQuery();

 return conn;
 }
 catch (Exception)
 {
 if (conn != null)
 {
 conn.Dispose();
 }
 throw;
 }
}

// ...

// Example query via ADO.NET SqlConnection.
// If row-level security is enabled, only Tenant 4's blogs are listed.
SqlDatabaseUtils.SqlRetryPolicy.ExecuteAction(() =>
{
 using (SqlConnection conn = OpenConnectionForTenant(
 sharding.ShardMap, tenantId4, connStrBldr.ConnectionString))
 {
 SqlCommand cmd = conn.CreateCommand();
 cmd.CommandText = @"SELECT * FROM Blogs";

 Console.WriteLine(@"--");
 All blogs for TenantId {0} (using ADO.NET SqlConnection):", tenantId4);

 SqlDataReader reader = cmd.ExecuteReader();
 while (reader.Read())
 {
 Console.WriteLine("{0}", reader["Name"]);
 }
 }
});

```

## 2. Data tier: Create row-level security policy

**Create a security policy to filter the rows each tenant can access**

Now that the application is setting SESSION\_CONTEXT with the current TenantId before querying, an RLS security policy can filter queries and exclude rows that have a different TenantId.

RLS is implemented in Transact-SQL. A user-defined function defines the access logic, and a security policy binds this function to any number of tables. For this project:

1. The function verifies that the application is connected to the database, and that the TenantId stored in the SESSION\_CONTEXT matches the TenantId of a given row.
  - The application is connected, rather than some other SQL user.
2. A FILTER predicate allows rows that meet the TenantId filter to pass through for SELECT, UPDATE, and DELETE queries.
  - A BLOCK predicate prevents rows that fail the filter from being INSERTed or UPDATED.
  - If SESSION\_CONTEXT has not been set, the function returns NULL, and no rows are visible or able to be inserted.

To enable RLS on all shards, execute the following T-SQL by using either Visual Studio (SSDT), SSMS, or the PowerShell script included in the project. Or if you are using [Elastic Database Jobs](#), you can automate execution of this T-SQL on all shards.

```
CREATE SCHEMA rls; -- Separate schema to organize RLS objects.
GO

CREATE FUNCTION rls.fn_tenantAccessPredicate(@TenantId int)
 RETURNS TABLE
 WITH SCHEMABINDING
AS
 RETURN SELECT 1 AS fn_accessResult
 -- Use the user in your application's connection string.
 -- Here we use 'dbo' only for demo purposes!
 WHERE DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('dbo')
 AND CAST(SESSION_CONTEXT(N'TenantId') AS int) = @TenantId;
GO

CREATE SECURITY POLICY rls.tenantAccessPolicy
 ADD FILTER PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Blogs,
 ADD BLOCK PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Blogs,
 ADD FILTER PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Posts,
 ADD BLOCK PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.Posts;
GO
```

#### TIP

In a complex project you might need to add the predicate on hundreds of tables, which could be tedious. There is a helper stored procedure that automatically generates a security policy, and adds a predicate on all tables in a schema. For more information, see the blog post at [Apply Row-Level Security to all tables - helper script \(blog\)](#).

Now if you run the sample application again, tenants see only rows that belong to them. In addition, the application cannot insert rows that belong to tenants other than the one currently connected to the shard database. Also, the app cannot update the TenantId in any rows it can see. If the app attempts to do either, a DbUpdateException is raised.

If you add a new table later, ALTER the security policy to add FILTER and BLOCK predicates on the new table.

```

ALTER SECURITY POLICY rls.tenantAccessPolicy
 ADD FILTER PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.MyNewTable,
 ADD BLOCK PREDICATE rls.fn_tenantAccessPredicate(TenantId) ON dbo.MyNewTable;
GO

```

## Add default constraints to automatically populate TenantId for INSERTs

You can put a default constraint on each table to automatically populate the TenantId with the value currently stored in SESSION\_CONTEXT when inserting rows. An example follows.

```

-- Create default constraints to auto-populate TenantId with the
-- value of SESSION_CONTEXT for inserts.
ALTER TABLE Blogs
 ADD CONSTRAINT df_TenantId_Blogs
 DEFAULT CAST(SESSION_CONTEXT(N'TenantId') AS int) FOR TenantId;
GO

ALTER TABLE Posts
 ADD CONSTRAINT df_TenantId_Posts
 DEFAULT CAST(SESSION_CONTEXT(N'TenantId') AS int) FOR TenantId;
GO

```

Now the application does not need to specify a TenantId when inserting rows:

```

SqlDatabaseUtils.SqlRetryPolicy.ExecuteAction(() =>
{
 using (var db = new ElasticScaleContext<int>(
 sharding.ShardMap, tenantId, connStrBldr.ConnectionString))
 {
 // The default constraint sets TenantId automatically!
 var blog = new Blog { Name = name };
 db.Blogs.Add(blog);
 db.SaveChanges();
 }
});

```

### NOTE

If you use default constraints for an Entity Framework project, it is recommended that you *NOT* include the TenantId column in your EF data model. This recommendation is because Entity Framework queries automatically supply default values that override the default constraints created in T-SQL that use SESSION\_CONTEXT. To use default constraints in the sample project, for instance, you should remove TenantId from DataClasses.cs (and run Add-Migration in the Package Manager Console) and use T-SQL to ensure that the field only exists in the database tables. This way, EF does automatically supply incorrect default values when inserting data.

## (Optional) Enable a *superuser* to access all rows

Some applications may want to create a *superuser* who can access all rows. A superuser could enable reporting across all tenants on all shards. Or a superuser could perform split-merge operations on shards that involve moving tenant rows between databases.

To enable a superuser, create a new SQL user (`superuser` in this example) in each shard database. Then alter the security policy with a new predicate function that allows this user to access all rows. Such a function is given next.

```

-- New predicate function that adds superuser logic.
CREATE FUNCTION rls.fn_tenantAccessPredicateWithSuperUser(@TenantId int)
 RETURNS TABLE
 WITH SCHEMABINDING
AS
 RETURN SELECT 1 AS fn_accessResult
 WHERE
 (
 DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('dbo') -- Replace 'dbo'.
 AND CAST(SESSION_CONTEXT(N'TenantId') AS int) = @TenantId
)
 OR
 (
 DATABASE_PRINCIPAL_ID() = DATABASE_PRINCIPAL_ID('superuser')
);
GO

-- Atomically swap in the new predicate function on each table.
ALTER SECURITY POLICY rls.tenantAccessPolicy
 ALTER FILTER PREDICATE rls.fn_tenantAccessPredicateWithSuperUser(TenantId) ON dbo.Blogs,
 ALTER BLOCK PREDICATE rls.fn_tenantAccessPredicateWithSuperUser(TenantId) ON dbo.Blogs,
 ALTER FILTER PREDICATE rls.fn_tenantAccessPredicateWithSuperUser(TenantId) ON dbo.Posts,
 ALTER BLOCK PREDICATE rls.fn_tenantAccessPredicateWithSuperUser(TenantId) ON dbo.Posts;
GO

```

## Maintenance

- **Adding new shards:** Execute the T-SQL script to enable RLS on any new shards, otherwise queries on these shards are not be filtered.
- **Adding new tables:** Add a FILTER and BLOCK predicate to the security policy on all shards whenever a new table is created. Otherwise queries on the new table are not be filtered. This addition can be automated by using a DDL trigger, as described in [Apply Row-Level Security automatically to newly created tables \(blog\)](#).

## Summary

Elastic database tools and row-level security can be used together to scale out an application's data tier with support for both multi-tenant and single-tenant shards. Multi-tenant shards can be used to store data more efficiently. This efficiency is pronounced where a large number of tenants have only a few rows of data. Single-tenant shards can support premium tenants which have stricter performance and isolation requirements. For more information, see [Row-Level Security reference](#).

## Additional resources

- [What is an Azure elastic pool?](#)
- [Scaling out with Azure SQL Database](#)
- [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#)
- [Authentication in multitenant apps, using Azure AD and OpenID Connect](#)
- [Tailspin Surveys application](#)

## Questions and Feature Requests

For questions, contact us on the [SQL Database forum](#). And add any feature requests to the [SQL Database feedback forum](#).

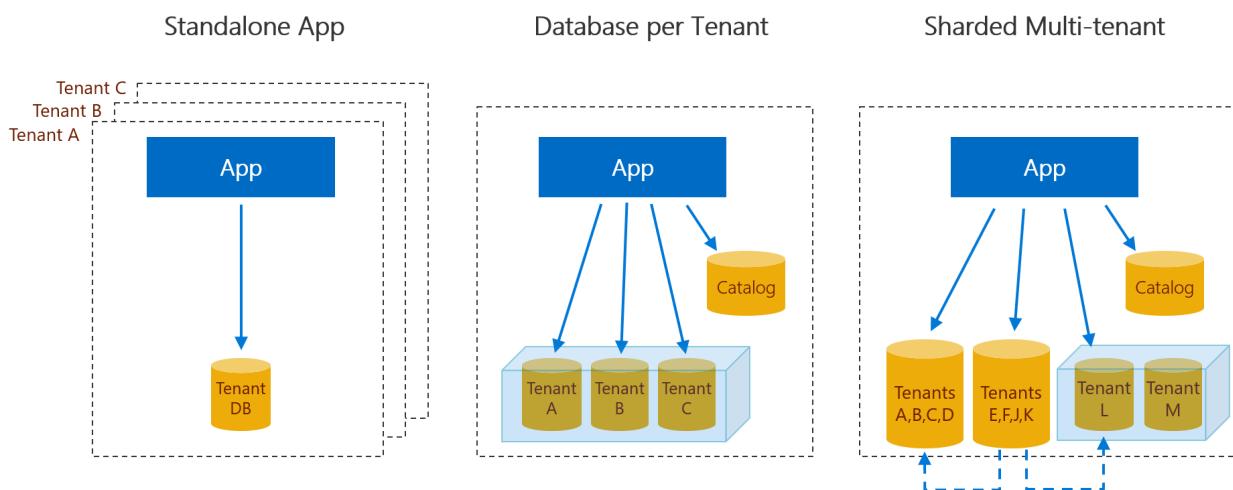
# The Wingtip Tickets SaaS application

11/7/2019 • 3 minutes to read • [Edit Online](#)

The same *Wingtip Tickets* SaaS application is implemented in each of three samples. The app is a simple event listing and ticketing SaaS app targeting small venues - theaters, clubs, etc. Each venue is a tenant of the app, and has its own data: venue details, lists of events, customers, ticket orders, etc. The app, together with the management scripts and tutorials, showcases an end-to-end SaaS scenario. This includes provisioning tenants, monitoring and managing performance, schema management, and cross-tenant reporting and analytics.

## Three SaaS application and tenancy patterns

Three versions of the app are available; each explores a different database tenancy pattern on Azure SQL Database. The first uses a standalone application per tenant with its own database. The second uses a multi-tenant app with a database per tenant. The third sample uses a multi-tenant app with sharded multi-tenant databases.



Each sample includes the application code, plus management scripts and tutorials that explore a range of design and management patterns. Each sample deploys in less than five minutes. All three can be deployed side-by-side so you can compare the differences in design and management.

## Standalone application per tenant pattern

The standalone app per tenant pattern uses a single tenant application with a database for each tenant. Each tenant's app, including its database, is deployed into a separate Azure resource group. The resource group can be deployed in the service provider's subscription or the tenant's subscription and managed by the provider on the tenant's behalf. The standalone app per tenant pattern provides the greatest tenant isolation, but is typically the most expensive as there's no opportunity to share resources between multiple tenants. This pattern is well suited to applications that might be more complex and which are deployed to smaller numbers of tenants. With standalone deployments, the app can be customized for each tenant more easily than in other patterns.

Check out the [tutorials](#) and code on GitHub [.../Microsoft/WingtipTicketsSaaS-StandaloneApp](https://github.com/Microsoft/WingtipTicketsSaaS-StandaloneApp).

## Database per tenant pattern

The database per tenant pattern is effective for service providers that are concerned with tenant isolation and want to run a centralized service that allows cost-efficient use of shared resources. A database is created for each venue, or tenant, and all the databases are centrally managed. Databases can be hosted in elastic pools to provide

cost-efficient and easy performance management, which leverages the unpredictable workload patterns of the tenants. A catalog database holds the mapping between tenants and their databases. This mapping is managed using the shard map management features of the [Elastic Database Client Library](#), which provides efficient connection management to the application.

Check out the [tutorials](#) and code on GitHub [../Microsoft/WingtipTicketsSaaS-DbPerTenant](#).

## Sharded multi-tenant database pattern

Multi-tenant databases are effective for service providers looking for lower cost per tenant and okay with reduced tenant isolation. This pattern allows packing large numbers of tenants into an individual database, driving the cost-per-tenant down. Near infinite scale is possible by sharding the tenants across multiple databases. A catalog database maps tenants to databases.

This pattern also allows a *hybrid* model in which you can optimize for cost with multiple tenants in a database, or optimize for isolation with a single tenant in their own database. The choice can be made on a tenant-by-tenant basis, either when the tenant is provisioned or later, with no impact on the application. This model can be used effectively when groups of tenants need to be treated differently. For example, low-cost tenants can be assigned to shared databases, while premium tenants can be assigned to their own databases.

Check out the [tutorials](#) and code on GitHub [../Microsoft/WingtipTicketsSaaS-MultiTenantDb](#).

## Next steps

### Conceptual descriptions

- A more detailed explanation of the application tenancy patterns is available at [Multi-tenant SaaS database tenancy patterns](#)

### Tutorials and code

- Standalone app per tenant:
  - [Tutorials for standalone app](#).
  - [Code for standalone app, on GitHub](#).
- Database per tenant:
  - [Tutorials for database per tenant](#).
  - [Code for database per tenant, on GitHub](#).
- Sharded multi-tenant:
  - [Tutorials for sharded multi-tenant](#).
  - [Code for sharded multi-tenant, on GitHub](#).

# General guidance for working with Wingtip Tickets sample SaaS apps

11/15/2019 • 3 minutes to read • [Edit Online](#)

This article contains general guidance for running the Wingtip Tickets sample SaaS applications that use Azure SQL Database.

## Download and unblock the Wingtip Tickets SaaS scripts

Executable contents (scripts, dlls) may be blocked by Windows when zip files are downloaded from an external source and extracted. When extracting the scripts from a zip file, **follow the steps below to unblock the .zip file before extracting**. This ensures the scripts are allowed to run.

1. Browse to the Wingtip Tickets SaaS GitHub repo for the database tenancy pattern you wish to explore:
  - [WingtipTicketsSaaS-StandaloneApp](#)
  - [WingtipTicketsSaaS-DbPerTenant](#)
  - [WingtipTicketsSaaS-MultiTenantDb](#)
2. Click **Clone or download**.
3. Click **Download zip** and save the file.
4. Right-click the zip file, and select **Properties**. The zip file name will correspond to the repo name. (ex. *WingtipTicketsSaaS-DbPerTenant-master.zip*)
5. On the **General** tab, select **Unblock**.
6. Click **OK**.
7. Extract the files.

Scripts are located in the ..\Learning Modules folder.

## Working with the Wingtip Tickets PowerShell scripts

To get the most out of the sample you need to dive into the provided scripts. Use breakpoints and step through the scripts as they execute and examine how the different SaaS patterns are implemented. To easily step through the provided scripts and modules for the best understanding, we recommend using the [PowerShell ISE](#).

### Update the configuration file for your deployment

Edit the **UserConfig.psm1** file with the resource group and user value that you set during deployment:

1. Open the *PowerShell ISE* and load ...\\Learning Modules\\UserConfig.psm1
2. Update *ResourceGroupName* and *Name* with the specific values for your deployment (on lines 10 and 11 only).
3. Save the changes!

Setting these values here simply keeps you from having to update these deployment-specific values in every script.

### Execute the scripts by pressing F5

Several scripts use *\$PSScriptRoot* to navigate folders, and *\$PSScriptRoot* is only evaluated when scripts are executed by pressing **F5**. Highlighting and running a selection (**F8**) can result in errors, so press **F5** when running scripts.

## Step through the scripts to examine the implementation

The best way to understand the scripts is by stepping through them to see what they do. Check out the included **Demo-** scripts that present an easy to follow high-level workflow. The **Demo-** scripts show the steps required to accomplish each task, so set breakpoints and drill deeper into the individual calls to see implementation details for the different SaaS patterns.

Tips for exploring and stepping through PowerShell scripts:

- Open **Demo-** scripts in the PowerShell ISE.
- Execute or continue with **F5** (using **F8** is not advised because `$PSScriptRoot` is not evaluated when running selections of a script).
- Place breakpoints by clicking or selecting a line and pressing **F9**.
- Step over a function or script call using **F10**.
- Step into a function or script call using **F11**.
- Step out of the current function or script call using **Shift + F11**.

## Explore database schema and execute SQL queries using SSMS

Use [SQL Server Management Studio \(SSMS\)](#) to connect and browse the application servers and databases.

The deployment initially has tenants and catalog SQL Database servers to connect to. The naming of the servers depends on the database tenancy pattern (see below for specifics).

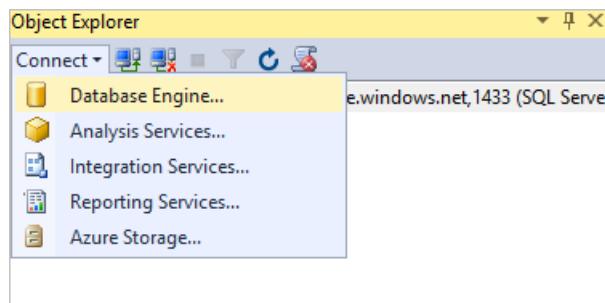
- **Standalone application:** servers for each tenant (ex. `contosoconcerthall-<User>` server) and `catalog-sa-<User>`
- **Database per tenant:** `tenants1-dpt-<User>` and `catalog-dpt-<User>` servers
- **Multi-tenant database:** `tenants1-mt-<User>` and `catalog-mt-<User>` servers

To ensure a successful demo connection, all servers have a [firewall rule](#) allowing all IPs through.

1. Open *SSMS* and connect to the tenants. The server name depends on the database tenancy pattern you've selected (see below for specifics):

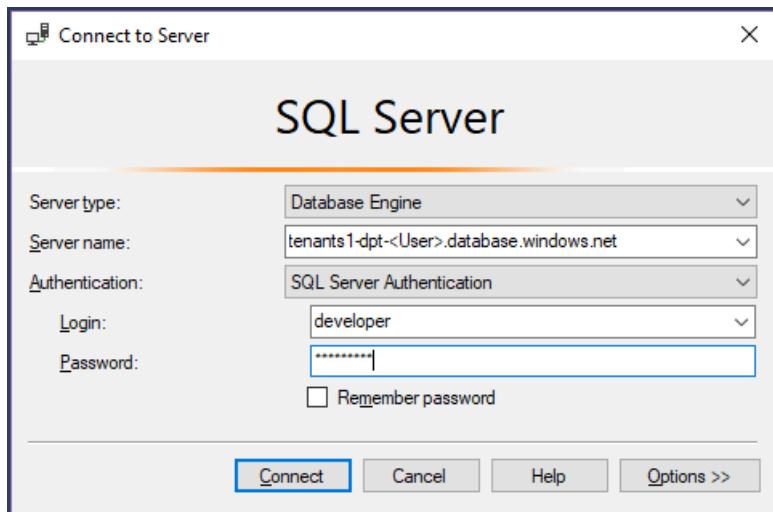
- **Standalone application:** servers of individual tenants (ex. `contosoconcerthall-<User>.database.windows.net`)
- **Database per tenant:** `tenants1-dpt-<User>.database.windows.net`
- **Multi-tenant database:** `tenants1-mt-<User>.database.windows.net`

2. Click **Connect > Database Engine...**:



3. Demo credentials are: Login = *developer*, Password = *P@ssword1*

The image below demonstrates the login for the *Database per tenant* pattern.

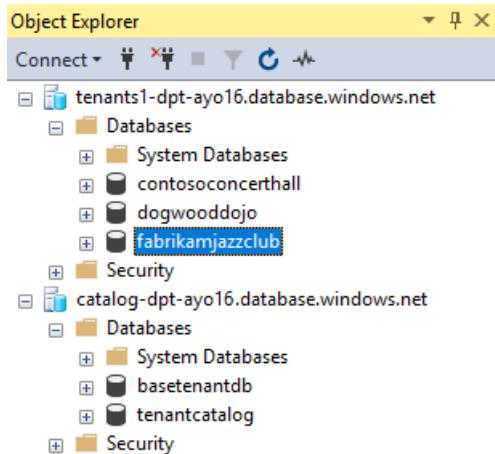


4. Repeat steps 2-3 and connect to the catalog server (see below for specific server names based on the database tenancy pattern selected)

- **Standalone application:** *catalog-sa-<User>.database.windows.net*
- **Database per tenant:** *catalog-dpt-<User>.database.windows.net*
- **Multi-tenant database:** *catalog-mt-<User>.database.windows.net*

After successfully connecting you should see all servers. Your list of databases might be different, depending on the tenants you have provisioned.

The image below demonstrates the log in for the *Database per tenant* pattern.



## Next steps

- [Deploy the Wingtip Tickets SaaS Standalone Application](#)
- [Deploy the Wingtip Tickets SaaS Database per Tenant application](#)
- [Deploy the Wingtip Tickets SaaS Multi-tenant Database application](#)

# Deploy and explore a standalone single-tenant application that uses Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

In this tutorial, you deploy and explore the Wingtip Tickets SaaS sample application developed using the standalone application, or app-per-tenant, pattern. The application is designed to showcase features of Azure SQL Database that simplify enabling multi-tenant SaaS scenarios.

The standalone application or app-per-tenant pattern deploys an application instance for each tenant. Each application is configured for a specific tenant and deployed in a separate Azure resource group. Multiple instances of the application are provisioned to provide a multi-tenant solution. This pattern is best suited to smaller numbers, of tenants where tenant isolation is a top priority. Azure has partner programs that allow resources to be deployed into a tenant's subscription and managed by a service provider on the tenant's behalf.

In this tutorial, you'll deploy three standalone applications for three tenants into your Azure subscription. You have full access to explore and work with the individual application components.

The application source code and management scripts are available in the [WingtipTicketsSaaS-StandaloneApp](#) GitHub repo. The application was created using Visual Studio 2015, and doesn't successfully open and compile in Visual Studio 2019 without updating.

In this tutorial you learn:

- How to deploy the Wingtip Tickets SaaS Standalone Application.
- Where to get the application source code, and management scripts.
- About the servers and databases that make up the app.

Additional tutorials will be released. They'll allow you to explore a range of management scenarios based on this application pattern.

## Deploy the Wingtip Tickets SaaS Standalone Application

Deploy the app for the three provided tenants:

1. Click each blue **Deploy to Azure** button to open the deployment template in the [Azure portal](#). Each template requires two parameter values; a name for a new resource group, and a user name that distinguishes this deployment from other deployments of the app. The next step provides details for setting these values.



**Contoso Concert Hall**



**Dogwood Dojo**



**Fabrikam Jazz Club**

2. Enter required parameter values for each deployment.

### **IMPORTANT**

Some authentication and server firewalls are intentionally unsecured for demonstration purposes. **Create a new resource group** for each application deployment. Do not use an existing resource group. Do not use this application, or any resources it creates, for production. Delete all the resource groups when you are finished with the applications to stop related billing.

It's best to use only lowercase letters, numbers, and hyphens in your resource names.

- For **Resource group**, select Create new, and then provide a lowercase Name for the resource group. **wingtip-sa-<venueName>-<user>** is the recommended pattern. For <venueName>, replace the venue name with no spaces. For <user>, replace the user value from below. With this pattern, resource group names might be *wingtip-sa-contosoconcerthall-af1*, *wingtip-sa-dogwooddojo-af1*, *wingtip-sa-fabrikamjazzclub-af1*.
- Select a **Location** from the drop-down list.
- For **User** - We recommend a short user value, such as your initials plus a digit: for example, *af1*.

### **3. Deploy the application.**

- Click to agree to the terms and conditions.
  - Click **Purchase**.
4. Monitor the status of all three deployments by clicking **Notifications** (the bell icon to the right of the search box). Deploying the apps takes around five minutes.

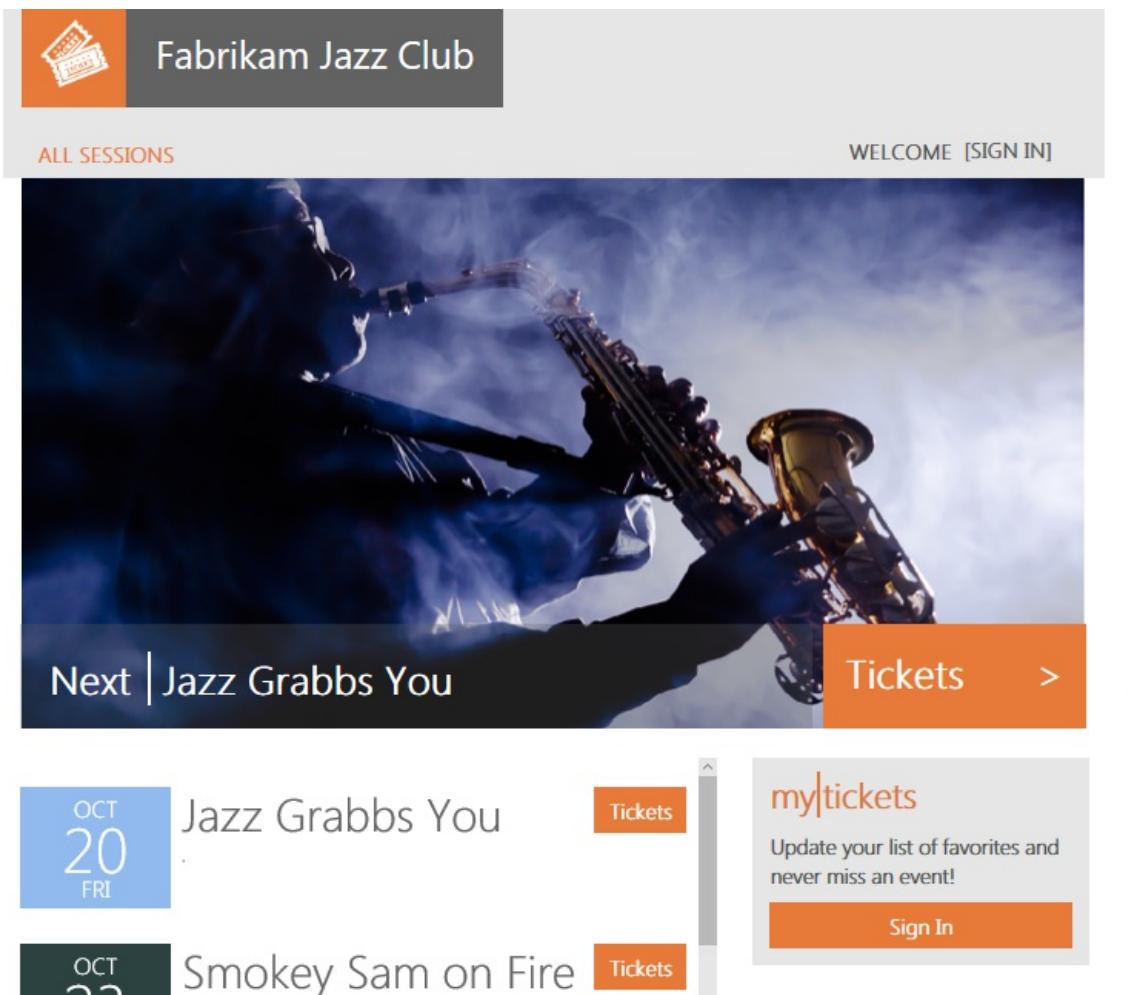
## Run the applications

The app showcases venues that host events. The venues are the tenants of the application. Each venue gets a personalized web site to list their events and sell tickets. Venue types include concert halls, jazz clubs, and sports clubs. In the sample, the type of venue determines the background photograph shown on the venue's web site. In the standalone app model, each venue has a separate application instance with its own standalone SQL database.

### 1. Open the events page for each of the three tenants in separate browser tabs:

- <http://events.contosoconcerthall.<user>.trafficmanager.net>
- <http://events.dogwooddojo.<user>.trafficmanager.net>
- <http://events.fabrikamjazzclub.<user>.trafficmanager.net>

(In each URL, replace <user> with your deployment's user value.)



To control the distribution of incoming requests, the app uses [Azure Traffic Manager](#). Each tenant-specific app instance includes the tenant name as part of the domain name in the URL. All the tenant URLs include your specific **User** value. The URLs follow the following format:

- <http://events.<venuename>.<user>.trafficmanager.net>

Each tenant's database **Location** is included in the app settings of the corresponding deployed app.

In a production environment, typically you create a CNAME DNS record to [point a company internet domain](#) to the URL of the traffic manager profile.

## Explore the servers and tenant databases

Let's look at some of the resources that were deployed:

1. In the [Azure portal](#), browse to the list of resource groups.
2. You should see the three tenant resource groups.
3. Open the **wingtip-sa-fabrikam-<user>** resource group, which contains the resources for the Fabrikam Jazz Club deployment. The **fabrikamjazzclub-<user>** server contains the **fabrikamjazzclub** database.

Each tenant database is a 50 DTU *standalone* database.

## Additional resources

- To learn about multi-tenant SaaS applications, see [Design patterns for multi-tenant SaaS applications](#).

## Delete resource groups to stop billing

When you have finished using the sample, delete all the resource groups you created to stop the associated billing.

## Next steps

In this tutorial you learned:

- How to deploy the Wingtip Tickets SaaS Standalone Application.
- About the servers and databases that make up the app.
- How to delete sample resources to stop related billing.

Next, try the [Provision and Catalog](#) tutorial in which you'll explore the use of a catalog of tenants that enables a range of cross-tenant scenarios such as schema management and tenant analytics.

# Provision and catalog new tenants using the application per tenant SaaS pattern

11/15/2019 • 7 minutes to read • [Edit Online](#)

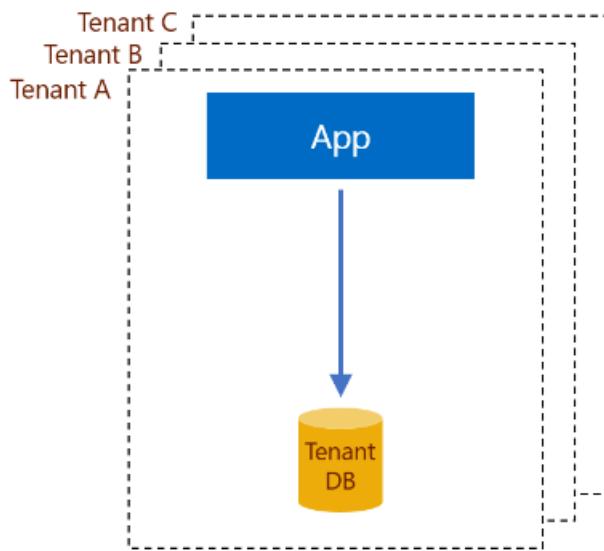
This article covers the provisioning and cataloging of new tenants using the standalone app per tenant SaaS pattern. This article has two major parts:

- Conceptual discussion of provisioning and cataloging new tenants
- A tutorial that highlights sample PowerShell code that accomplishes the provisioning and cataloging
  - The tutorial uses the Wingtip Tickets sample SaaS application, adapted to the standalone app per tenant pattern.

## Standalone application per tenant pattern

The standalone app per tenant pattern is one of several patterns for multi-tenant SaaS applications. In this pattern, a standalone app is provisioned for each tenant. The application comprises application level components and a SQL database. Each tenant app can be deployed in the vendor's subscription. Alternatively, Azure offers a [managed applications program](#) in which an app can be deployed in a tenant's subscription and managed by the vendor on the tenant's behalf.

### Standalone app per tenant

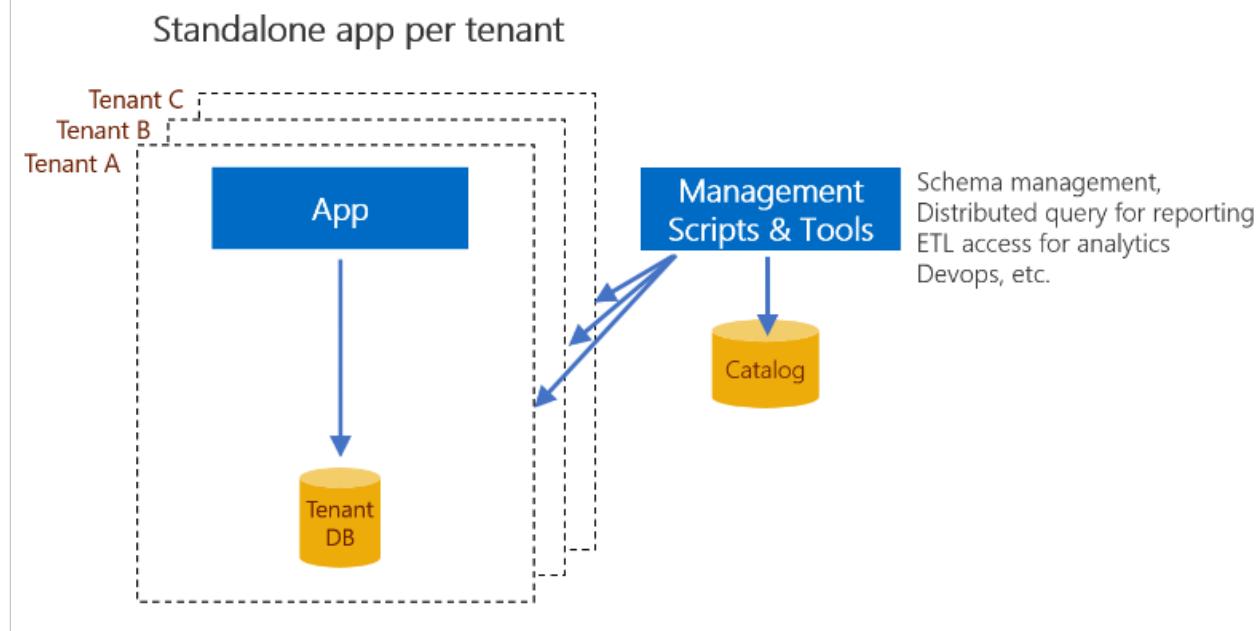


When deploying an application for a tenant, the app and database are provisioned in a new resource group created for the tenant. Using separate resource groups isolates each tenant's application resources and allows them to be managed independently. Within each resource group, each application instance is configured to access its corresponding database directly. This connection model contrasts with other patterns that use a catalog to broker connections between the app and the database. And as there is no resource sharing, each tenant database must be provisioned with sufficient resources to handle its peak load. This pattern tends to be used for SaaS applications with fewer tenants, where there is a strong emphasis on tenant isolation and less emphasis on resource costs.

## Using a tenant catalog with the application per tenant pattern

While each tenant's app and database are fully isolated, various management and analytics scenarios may operate

across tenants. For example, applying a schema change for a new release of the application requires changes to the schema of each tenant database. Reporting and analytics scenarios may also require access to all the tenant databases regardless of where they are deployed.



The tenant catalog holds a mapping between a tenant identifier and a tenant database, allowing an identifier to be resolved to a server and database name. In the Wingtip SaaS app, the tenant identifier is computed as a hash of the tenant name, although other schemes could be used. While standalone applications don't need the catalog to manage connections, the catalog can be used to scope other actions to a set of tenant databases. For example, Elastic Query can use the catalog to determine the set of databases across which queries are distributed for cross-tenant reporting.

## Elastic Database Client Library

In the Wingtip sample application, the catalog is implemented by the shard management features of the [Elastic Database Client Library](#) (EDCL). The library enables an application to create, manage, and use a shard map that is stored in a database. In the Wingtip Tickets sample, the catalog is stored in the *tenant catalog* database. The shard maps a tenant key to the shard (database) in which that tenant's data is stored. EDCL functions manage a *global shard map* stored in tables in the *tenant catalog* database and a *local shard map* stored in each shard.

EDCL functions can be called from applications or PowerShell scripts to create and manage the entries in the shard map. Other EDCL functions can be used to retrieve the set of shards or connect to the correct database for given tenant key.

#### IMPORTANT

Do not edit the data in the catalog database or the local shard map in the tenant databases directly. Direct updates are not supported due to the high risk of data corruption. Instead, edit the mapping data by using EDCL APIs only.

## Tenant provisioning

Each tenant requires a new Azure resource group, which must be created before resources can be provisioned within it. Once the resource group exists, an Azure Resource Management template can be used to deploy the application components and the database, and then configure the database connection. To initialize the database schema, the template can import a bacpac file. Alternatively, the database can be created as a copy of a 'template' database. The database is then further updated with initial venue data and registered in the catalog.

# Tutorial

In this tutorial you learn how to:

- Provision a catalog
- Register the sample tenant databases that you deployed earlier in the catalog
- Provision an additional tenant and register it in the catalog

An Azure Resource Manager template is used to deploy and configure the application, create the tenant database, and then import a bacpac file to initialize it. The import request may be queued for several minutes before it is actioned.

At the end of this tutorial, you have a set of standalone tenant applications, with each database registered in the catalog.

## Prerequisites

To complete this tutorial, make sure the following prerequisites are completed:

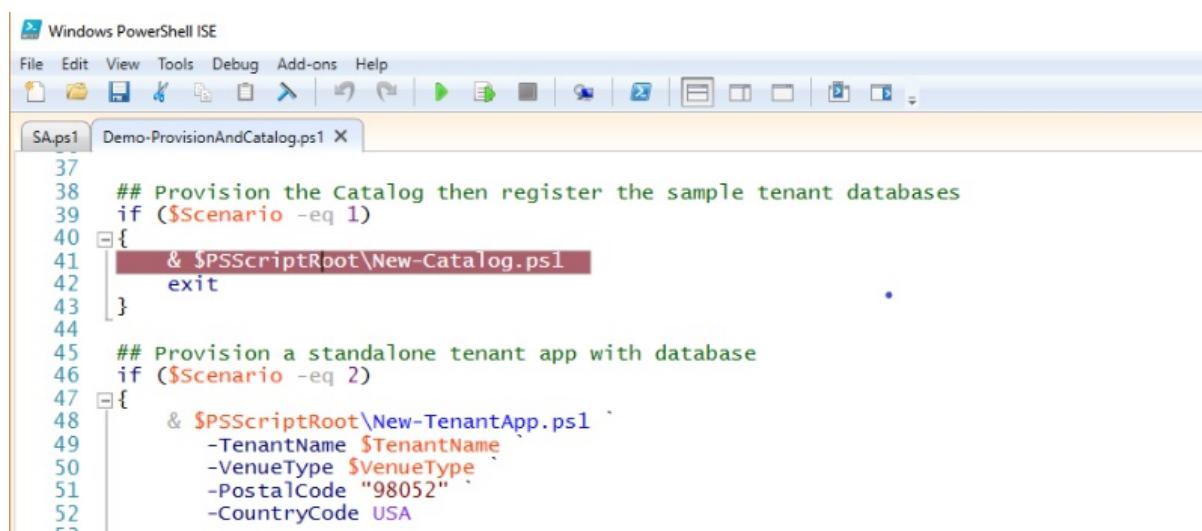
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- The three sample tenant apps are deployed. To deploy these apps in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Standalone Application pattern](#).

## Provision the catalog

In this task, you learn how to provision the catalog used to register all the tenant databases. You will:

- **Provision the catalog database** using an Azure resource management template. The database is initialized by importing a bacpac file.
- **Register the sample tenant apps** that you deployed earlier. Each tenant is registered using a key constructed from a hash of the tenant name. The tenant name is also stored in an extension table in the catalog.

1. In PowerShell ISE, open ...\\Learning Modules\\UserConfig.psm and update the <user> value to the value you used when deploying the three sample applications. **Save the file**.
2. In PowerShell ISE, open ...\\Learning Modules\\ProvisionTenants\\Demo-ProvisionAndCatalog.ps1 and set **\$Scenario = 1**. Deploy the tenant catalog and register the pre-defined tenants.
3. Add a breakpoint by putting your cursor anywhere on the line that says, `& $PSScriptRoot\\New-Catalog.ps1`, and then press **F9**.



```
Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
S:\aps1 Demo-ProvisionAndCatalog.ps1 X
37
38 ## Provision the Catalog then register the sample tenant databases
39 if ($Scenario -eq 1)
40 {
41 & $PSScriptRoot\New-Catalog.ps1
42 exit
43 }
44
45 ## Provision a standalone tenant app with database
46 if ($Scenario -eq 2)
47 {
48 & $PSScriptRoot\New-TenantApp.ps1
49 -TenantName $TenantName
50 -VenueType $VenueType
51 -PostalCode "98052"
52 -CountryCode USA
```

4. Run the script by pressing **F5**.

5. After script execution stops at the breakpoint, press **F11** to step into the New-Catalog.ps1 script.
  6. Trace the script's execution using the Debug menu options, F10 and F11, to step over or into called functions.
    - For more information about debugging PowerShell scripts, see [Tips on working with and debugging PowerShell scripts](#).
- Once the script completes, the catalog will exist and all the sample tenants will be registered.
- Now look at the resources you created.
1. Open the [Azure portal](#) and browse the resource groups. Open the **wingtip-sa-catalog-<user>** resource group and note the catalog server and database.
  2. Open the database in the portal and select *Data explorer* from the left-hand menu. Click the Login command and then enter the Password = **P@ssword1**.
  3. Explore the schema of the *tenantcatalog* database.
    - The objects in the `__ShardManagement` schema are all provided by the Elastic Database Client Library.
    - The `Tenants` table and `TenantsExtended` view are extensions added in the sample that demonstrate how you can extend the catalog to provide additional value.
  4. Run the query, `SELECT * FROM dbo.TenantsExtended`.

TENANTNAME	SERVICEPLAN	SERVERNAME	DATABASENAME
Contoso Concert Hall	standard	contosoconehmerhall-bq1.database.windows.net	contosoconehmerhall
Dogwood Dojo	standard	dogwooddojo-bq1.database.windows.net	dogwooddojo
Fabrikam Jazz Club	standard	fabrikamjazzclub-bq1.database.windows.net	fabrikamjazzclub
Red Maple Racing	standard	redmapleracing-bq1.database.windows.net	redmapleracing

As an alternative to using the Data Explorer you can connect to the database from SQL Server Management Studio. To do this, connect to the server wingtip-

Note that you should not edit data directly in the catalog - always use the shard management APIs.

## Provision a new tenant application

In this task, you learn how to provision a single tenant application. You will:

- **Create a new resource group** for the tenant.
- **Provision the application and database** into the new resource group using an Azure resource management template. This action includes initializing the database with common schema and reference data by importing a bacpac file.
- **Initialize the database with basic tenant information.** This action includes specifying the venue type, which determines the photograph used as the background on its events web site.
- **Register the database in the catalog database.**

1. In PowerShell ISE, open ...\\Learning Modules\\ProvisionTenants\\Demo-ProvisionAndCatalog.ps1 and set \$Scenario = 2. Deploy the tenant catalog and register the pre-defined tenants
2. Add a breakpoint in the script by putting your cursor anywhere on line 49 that says, & \$PSScriptRoot\\New-TenantApp.ps1, and then press **F9**.
3. Run the script by pressing **F5**.
4. After script execution stops at the breakpoint, press **F11** to step into the New-Catalog.ps1 script.
5. Trace the script's execution using the Debug menu options, F10 and F11, to step over or into called functions.

After the tenant has been provisioned, the new tenant's events website is opened.



The screenshot shows a dynamic image of a white and green Formula 1 racing car in motion, blurred background suggesting speed. At the top left is a logo with two tickets and the text "Red Maple Racing". To the right is a "WELCOME [SIGN IN]" button. Below the image are navigation links: "ALL RACES" on the left and "Tickets" with a right-pointing arrow on the right. At the bottom left, there's a calendar entry for "FEB 5 MON" followed by "Event 3" and "Performer 3". At the bottom right, there's another "Tickets" button and a "my|tickets" section with a "Sign In" button.

Date	Event	Performer	Tickets
FEB 5 MON	Event 3	Performer 3	<a href="#">Tickets</a>
FEB 6 TUE	Event 4		<a href="#">Tickets</a>

You can then inspect the new resources created in the Azure portal.

The screenshot shows the Azure portal interface for a specific resource group. The left sidebar contains various navigation links such as Overview, Activity log, Access control (IAM), Tags, SETTINGS, Quickstart, Resource costs, Deployments, Policies, Properties, Locks, and Automation script. The main content area displays subscription details (Subscription ID: 112900e0-ad0e-4ab8-aea2-b09a07253f30, Wingtip SaaS - Microsoft Azure Sponsorship) and deployment status (1 Succeeded). Below this, there is a list of 5 items, each with a checkbox, a name, a type, and a location. The items are:

	NAME	TYPE	LOCATION
<input type="checkbox"/>	events-redmapleracing-bg1	Traffic Manager profile	global
<input type="checkbox"/>	events-redmapleracing-bg1	App Service plan	South Central US
<input type="checkbox"/>	events-redmapleracing-bg1	App Service	South Central US
<input type="checkbox"/>	redmapleracing-bg1	SQL server	South Central US
<input type="checkbox"/>	redmapleracing	SQL database	South Central US

## To stop billing, delete resource groups

When you have finished exploring the sample, delete all the resource groups you created to stop the associated billing.

## Additional resources

- To learn more about multi-tenant SaaS database applications, see [Design patterns for multi-tenant SaaS applications](#).

## Next steps

In this tutorial you learned:

- How to deploy the Wingtip Tickets SaaS Standalone Application.
- About the servers and databases that make up the app.
- How to delete sample resources to stop related billing.

You can explore how the catalog is used to support various cross-tenant scenarios using the database-per-tenant version of the [Wingtip Tickets SaaS application](#).

# Introduction to a multitenant SaaS app that uses the database-per-tenant pattern with SQL Database

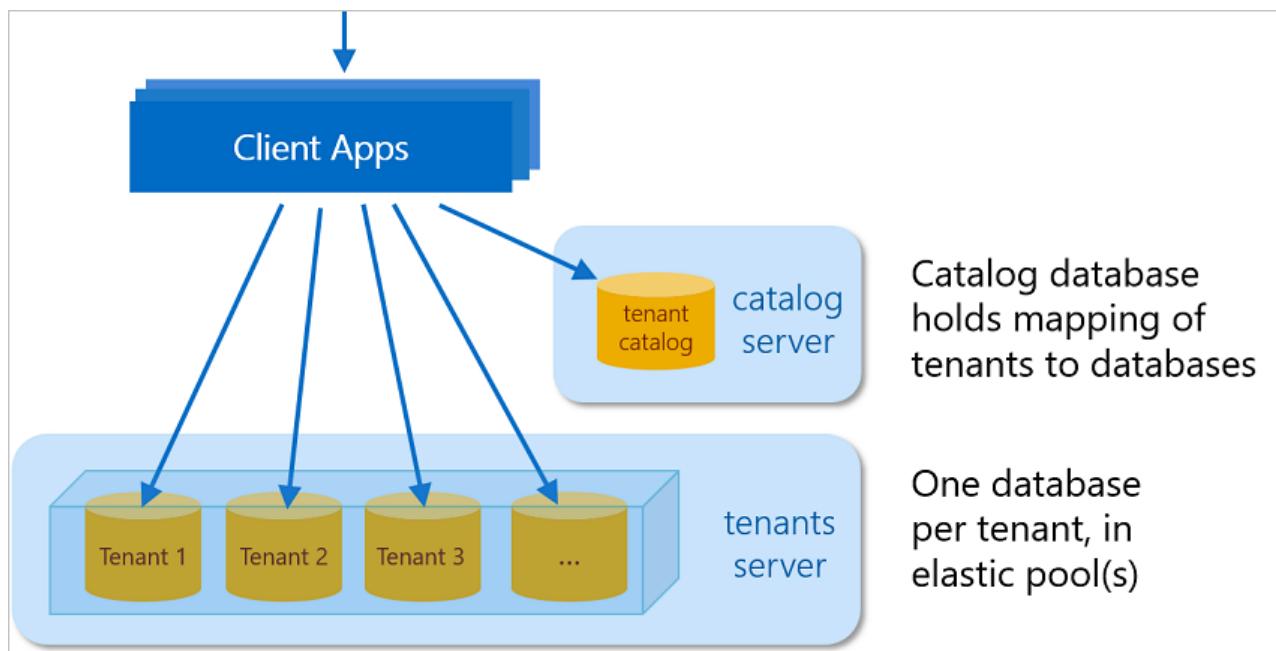
11/7/2019 • 2 minutes to read • [Edit Online](#)

The Wingtip SaaS application is a sample multitenant app. The app uses the database-per-tenant SaaS application pattern to service multiple tenants. The app showcases features of Azure SQL Database that enable SaaS scenarios by using several SaaS design and management patterns. To quickly get up and running, the Wingtip SaaS app deploys in less than five minutes.

Application source code and management scripts are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. Before you start, see the [general guidance](#) for steps to download and unblock the Wingtip Tickets management scripts.

## Application architecture

The Wingtip SaaS app uses the database-per-tenant model. It uses SQL elastic pools to maximize efficiency. For provisioning and mapping tenants to their data, a catalog database is used. The core Wingtip SaaS application uses a pool with three sample tenants, plus the catalog database. The catalog and tenant servers have been provisioned with DNS aliases. These aliases are used to maintain a reference to the active resources used by the Wingtip application. These aliases are updated to point to recovery resources in the disaster recovery tutorials. Completing many of the Wingtip SaaS tutorials results in add-ons to the initial deployment. Add-ons such as analytic databases and cross-database schema management are introduced.



As you go through the tutorials and work with the app, focus on the SaaS patterns as they relate to the data tier. In other words, focus on the data tier, and don't overanalyze the app itself. Understanding the implementation of these SaaS patterns is key to implementing these patterns in your applications. Also consider any necessary modifications for your specific business requirements.

## SQL Database Wingtip SaaS tutorials

After you deploy the app, explore the following tutorials that build on the initial deployment. These tutorials explore common SaaS patterns that take advantage of built-in features of SQL Database, Azure SQL Data

Warehouse, and other Azure services. Tutorials include PowerShell scripts with detailed explanations. The explanations simplify understanding and implementation of the same SaaS management patterns in your applications.

TUTORIAL	DESCRIPTION
<a href="#">Guidance and tips for the SQL Database multitenant SaaS app example</a>	Download and run PowerShell scripts to prepare parts of the application.
<a href="#">Deploy and explore the Wingtip SaaS application</a>	Deploy and explore the Wingtip SaaS application with your Azure subscription.
<a href="#">Provision and catalog tenants</a>	Learn how the application connects to tenants by using a catalog database, and how the catalog maps tenants to their data.
<a href="#">Monitor and manage performance</a>	Learn how to use monitoring features of SQL Database and set alerts when performance thresholds are exceeded.
<a href="#">Monitor with Azure Monitor logs</a>	Learn how to use <a href="#">Azure Monitor logs</a> to monitor large amounts of resources across multiple pools.
<a href="#">Restore a single tenant</a>	Learn how to restore a tenant database to a prior point in time. Also learn how to restore to a parallel database, which leaves the existing tenant database online.
<a href="#">Manage tenant database schema</a>	Learn how to update schema and update reference data across all tenant databases.
<a href="#">Run cross-tenant distributed queries</a>	Create an ad hoc analytics database, and run real-time distributed queries across all tenants.
<a href="#">Run analytics on extracted tenant data</a>	Extract tenant data into an analytics database or data warehouse for offline analytics queries.

## Next steps

- General guidance and tips when you deploy and use the Wingtip Tickets SaaS app example
- Deploy the Wingtip SaaS application

# Deploy and explore a multitenant SaaS app that uses the database-per-tenant pattern with SQL Database

11/7/2019 • 10 minutes to read • [Edit Online](#)

In this tutorial, you deploy and explore the Wingtip Tickets SaaS database-per-tenant application (Wingtip). The app uses a database-per-tenant pattern to store the data of multiple tenants. The app is designed to showcase features of Azure SQL Database that simplify how to enable SaaS scenarios.

Five minutes after you select **Deploy to Azure**, you have a multitenant SaaS application. The app includes a SQL database that runs in the cloud. The app is deployed with three sample tenants, each with its own database. All the databases are deployed into a SQL elastic pool. The app is deployed to your Azure subscription. You have full access to explore and work with the individual components of the app. The application C# source code and the management scripts are available in the [WingtipTicketsSaaS-DbPerTenant GitHub repo](#).

In this tutorial you learn:

- How to deploy the Wingtip SaaS application.
- Where to get the application source code and management scripts.
- About the servers, pools, and databases that make up the app.
- How tenants are mapped to their data with the *catalog*.
- How to provision a new tenant.
- How to monitor tenant activity in the app.

A [series of related tutorials](#) offers to explore various SaaS design and management patterns. The tutorials build beyond this initial deployment. When you use the tutorials, you can examine the provided scripts to see how the different SaaS patterns are implemented. The scripts demonstrate how features of SQL Database simplify the development of SaaS applications.

## Prerequisites

To complete this tutorial, make sure Azure PowerShell is installed. For more information, see [Get started with Azure PowerShell](#).

## Deploy the Wingtip Tickets SaaS application

### Plan the names

In the steps of this section, you provide a user value that is used to make sure resource names are globally unique. You also provide a name for the resource group that contains all the resources created by a deployment of the app. For a fictitious person named Ann Finley, we suggest:

- **User:** *af1* is made up of Ann Finley's initials plus a digit. If you deploy the app a second time, use a different value. An example is *af2*.
- **Resource group:** *wingtip-dpt-af1* indicates this is the database-per-tenant app. Append the user name *af1* to correlate the resource group name with the names of the resources it contains.

Choose your names now, and write them down.

### Steps

1. To open the Wingtip Tickets SaaS database-per-tenant deployment template in the Azure portal, select **Deploy to Azure**.

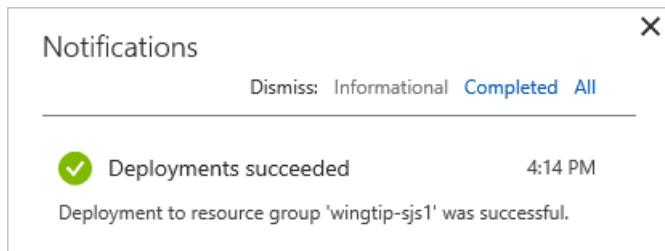


2. Enter values in the template for the required parameters.

**IMPORTANT**

Some authentication and server firewalls are intentionally unsecured for demonstration purposes. We recommend that you create a new resource group. Don't use existing resource groups, servers, or pools. Don't use this application, scripts, or any deployed resources for production. Delete this resource group when you're finished with the application to stop related billing.

- **Resource group:** Select **Create new**, and provide the unique name you chose earlier for the resource group.
  - **Location:** Select a location from the drop-down list.
  - **User:** Use the user name value you chose earlier.
3. Deploy the application.
    - a. Select to agree to the terms and conditions.
    - b. Select **Purchase**.
  4. To monitor deployment status, select **Notifications** (the bell icon to the right of the search box). Deploying the Wingtip Tickets SaaS app takes approximately five minutes.



## Download and unblock the Wingtip Tickets management scripts

While the application deploys, download the source code and management scripts.

**IMPORTANT**

Executable contents (scripts and DLLs) might be blocked by Windows when .zip files are downloaded from an external source and extracted. Follow the steps to unblock the .zip file before you extract the scripts. Unblocking makes sure the scripts are allowed to run.

1. Browse to the [WingtipTicketsSaaS-DbPerTenant GitHub repo](#).
2. Select **Clone or download**.
3. Select **Download ZIP**, and then save the file.
4. Right-click the **WingtipTicketsSaaS-DbPerTenant-master.zip** file, and then select **Properties**.
5. On the **General** tab, select **Unblock > Apply**.
6. Select **OK**, and extract the files

Scripts are located in the ...\\WingtipTicketsSaaS-DbPerTenant-master\\Learning Modules folder.

# Update the user configuration file for this deployment

Before you run any scripts, update the resource group and user values in the User Config file. Set these variables to the values you used during deployment.

1. In the PowerShell ISE, open ...\\Learning Modules\\**UserConfig.psm1**
2. Update **ResourceGroupName** and **Name** with the specific values for your deployment (on lines 10 and 11 only).
3. Save the changes.

These values are referenced in nearly every script.

## Run the application

The app showcases venues that host events. Venue types include concert halls, jazz clubs, and sports clubs. In Wingtip Tickets, venues are registered as tenants. Being a tenant gives a venue an easy way to list events and to sell tickets to their customers. Each venue gets a personalized website to list their events and to sell tickets.

Internally in the app, each tenant gets a SQL database deployed into a SQL elastic pool.

A central **Events Hub** page provides a list of links to the tenants in your deployment.

1. Use the URL to open the Events Hub in your web browser: <http://events.wingtip-dpt.<user>.trafficmanager.net>. Substitute <user> with your deployment's user value.

The screenshot shows the 'Events Hub' section of the Wingtip Tickets Platform. At the top, there's a search bar and a list of venues under the heading 'Venues'. The list includes 'Contoso Concert Hall', 'Dogwood Dojo', and 'Fabrikam Jazz Club'. Below this, there's a footer with the platform logo and a note about running on Azure SQL Database. To the right, there's a link to learn how to build a SaaS app on SQL Database.

Wingtip Tickets Platform

Events Hub

WELCOME

Welcome to the Wingtip Tickets Platform! Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database.

search

Venues

Contoso Concert Hall

Dogwood Dojo

Fabrikam Jazz Club

1 | End

LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE

Running on Azure SQL Database. © 2017 Microsoft

server: catalog gmt100wtp.database.windows.net

database: tenantcatalog

Wingtip Tickets Platform

Running on Azure SQL Database

2. Select **Fabrikam Jazz Club** in the Events Hub.



**Jazz Grabbs You**

**Smokey Sam on Fire**

**mytickets**

Update your list of favorites and never miss an event!

Sign In

## Azure Traffic Manager

The Wingtip application uses [Azure Traffic Manager](#) to control the distribution of incoming requests. The URL to access the events page for a specific tenant uses the following format:

- <http://events.wingtip-dpt.<user>.trafficmanager.net/fabrikamjazzclub>

The parts of the preceding format are explained in the following table.

URL PART	DESCRIPTION
<a href="http://events.wingtip-dpt">http://events.wingtip-dpt</a>	The events parts of the Wingtip app.  - <i>dpt</i> distinguishes the <i>database-per-tenant</i> implementation of Wingtip Tickets from other implementations. Examples are the <i>single app-per-tenant</i> (- <i>sa</i> ) or <i>multitenant database</i> (- <i>mt</i> ) implementations.
.< <i>user</i> >	<i>af1</i> in the example.
.trafficmanager.net/	Traffic Manager, base URL.
fabrikamjazzclub	Identifies the tenant named Fabrikam Jazz Club.

- The tenant name is parsed from the URL by the events app.
- The tenant name is used to create a key.
- The key is used to access the catalog to obtain the location of the tenant's database.

- The catalog is implemented by using *shard map management*.
- The Events Hub uses extended metadata in the catalog to construct the list-of-events page URLs for each tenant.

In a production environment, typically you create a CNAME DNS record to *point a company internet domain* to the Traffic Manager DNS name.

#### **NOTE**

It may not be immediately obvious what the use of the traffic manager is in this tutorial. The goal of this series of tutorials is to showcase patterns that can handle the scale of a complex production environment. In such a case, for example, you would have multiple web apps distributed across the globe, co-located with databases and you would need traffic manager to route between these instances. Another set of tutorials that illustrates the use of traffic manager though are the [geo-restore](#) and the [geo-replication](#) tutorials. In these tutorials, traffic manager is used to help to switch over to a recovery instance of the SaaS app in the event of a regional outage.

## Start generating load on the tenant databases

Now that the app is deployed, let's put it to work.

The *Demo-LoadGenerator* PowerShell script starts a workload that runs against all tenant databases. The real-world load on many SaaS apps is sporadic and unpredictable. To simulate this type of load, the generator produces a load with randomized spikes or bursts of activity on each tenant. The bursts occur at randomized intervals. It takes several minutes for the load pattern to emerge. Let the generator run for at least three or four minutes before you monitor the load.

1. In the PowerShell ISE, open the ...\\Learning Modules\\Utilities\\*Demo-LoadGenerator.ps1* script.
2. Press F5 to run the script and start the load generator. Leave the default parameter values for now.
3. Sign in to your Azure account, and select the subscription you want to use, if necessary.

The load generator script starts a background job for each database in the catalog and then stops. If you rerun the load generator script, it stops any background jobs that are running before it starts new ones.

### Monitor the background jobs

If you want to control and monitor the background jobs, use the following cmdlets:

- `Get-Job`
- `Receive-Job`
- `Stop-Job`

### **Demo-LoadGenerator.ps1 actions**

*Demo-LoadGenerator.ps1* mimics an active workload of customer transactions. The following steps describe the sequence of actions that *Demo-LoadGenerator.ps1* initiates:

1. *Demo-LoadGenerator.ps1* starts *LoadGenerator.ps1* in the foreground.
  - Both .ps1 files are stored under the folders Learning Modules\\Utilities\\.
2. *LoadGenerator.ps1* loops through all tenant databases in the catalog.
3. *LoadGenerator.ps1* starts a background PowerShell job for each tenant database:
  - By default, the background jobs run for 120 minutes.
  - Each job causes a CPU-based load on one tenant database by executing `sp_CpuLoadGenerator`. The intensity and duration of the load varies depending on `$DemoScenario`.
  - `sp_CpuLoadGenerator` loops around a SQL SELECT statement that causes a high CPU load. The time

interval between issues of the SELECT varies according to parameter values to create a controllable CPU load. Load levels and intervals are randomized to simulate more realistic loads.

- This .sql file is stored under *WingtipTenantDB\dbo\StoredProcedures*.
4. If `$OneTime = $false`, the load generator starts the background jobs and then continues to run. Every 10 seconds, it monitors for any new tenants that are provisioned. If you set `$OneTime = $true`, the LoadGenerator starts the background jobs and then stops running in the foreground. For this tutorial, leave `$OneTime = $false`.

Use Ctrl-C or Stop Operation Ctrl-Break if you want to stop or restart the load generator.

If you leave the load generator running in the foreground, use another PowerShell ISE instance to run other PowerShell scripts.

Before you continue with the next section, leave the load generator running in the job-invoking state.

## Provision a new tenant

The initial deployment creates three sample tenants. Now you create another tenant to see the impact on the deployed application. In the Wingtip app, the workflow to provision new tenants is explained in the [Provision and catalog tutorial](#). In this phase, you create a new tenant, which takes less than one minute.

1. Open a new PowerShell ISE.
2. Open ...\\Learning Modules\\Provision and Catalog\\*Demo-ProvisionAndCatalog.ps1*.
3. To run the script, press F5. Leave the default values for now.

### NOTE

Many Wingtip SaaS scripts use `$PSScriptRoot` to browse folders to call functions in other scripts. This variable is evaluated only when the full script is executed by pressing F5. Highlighting and running a selection with F8 can result in errors. To run the scripts, press F5.

The new tenant database is:

- Created in an SQL elastic pool.
- Initialized.
- Registered in the catalog.

After successful provisioning, the *Events* site of the new tenant appears in your browser.

The screenshot shows a racing-themed website. At the top left is a logo with two orange tickets. To its right is the text "Red Maple Racing". On the far left, there's a link "ALL RACES". On the far right, it says "WELCOME [SIGN IN]". Below the header is a large image of a race car cockpit from a low angle, showing the driver's helmet and the steering wheel. Below this image is a dark banner with the text "Next | Event 3" on the left and "Tickets >" on the right. Underneath this banner, there are two event cards. The first card, for "Event 3" on "MAY 7 SUN", includes "Performer 3" and a "Tickets" button. The second card, for "Event 4" on "MAY 10", also has a "Tickets" button. To the right of these cards is a "my|tickets" section with the text "Update your list of favorites and never miss an event!" and a "Sign In" button.

Date	Event	Performer	Tickets
MAY 7 SUN	Event 3	Performer 3	<a href="#">Tickets</a>
MAY 10	Event 4		<a href="#">Tickets</a>

Refresh the Events Hub to make the new tenant appear in the list.

## Explore the servers, pools, and tenant databases

Now that you've started running a load against the collection of tenants, let's look at some of the resources that were deployed.

1. In the [Azure portal](#), browse to your list of SQL servers. Then open the **catalog-dpt-<USER>** server.
  - The catalog server contains two databases, **tenantcatalog** and **basetenantdb** (a template database that's copied to create new tenants).

Microsoft Azure SQL servers > catalog-sjs1

Report a bug Search resources

catalog-sjs1

+ New database + New pool Import database Reset password → Move Delete

Search (Ctrl+ /)

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

SETTINGS Quick start Firewall Long-term backup retention Auditing & Threat Detection Active Directory admin Deleted databases Properties Locks Automation script

SUPPORT + TROUBLESHOOTING

Resource group (change) wingtip-sjs1 Status Available Location Central US Subscription name (change) SQL DB Content Subscription ID

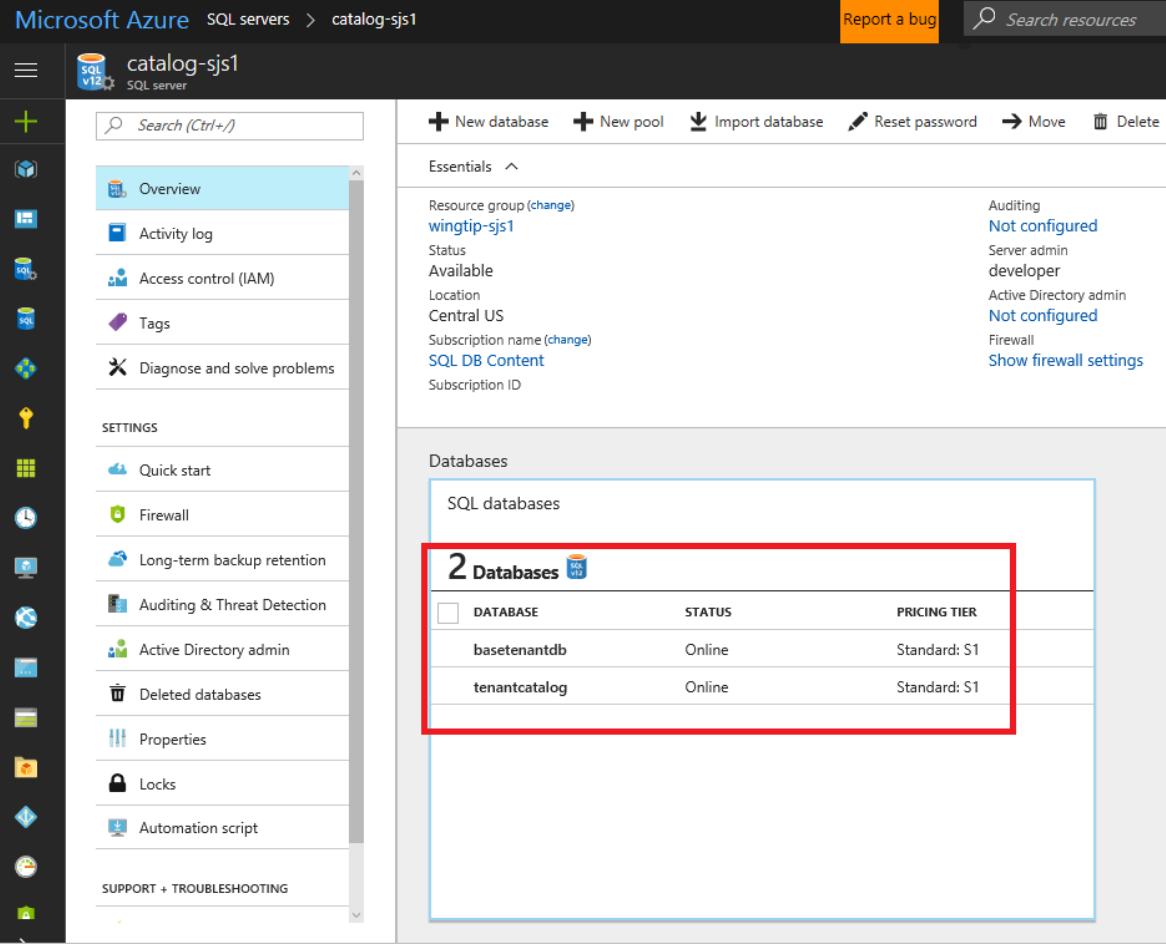
Auditing Not configured Server admin developer Active Directory admin Not configured Firewall Show firewall settings

Databases

SQL databases

2 Databases

	DATABASE	STATUS	PRICING TIER
<input type="checkbox"/>	basetenantdb	Online	Standard: S1
<input type="checkbox"/>	tenantcatalog	Online	Standard: S1



2. Go back to your list of SQL servers.
3. Open the **tenants1-dpt-<USER>** server that holds the tenant databases.
4. See the following items:
  - Each tenant database is an **Elastic Standard** database in a 50-eDTU standard pool.
  - The Red Maple Racing database is the tenant database you provisioned previously.

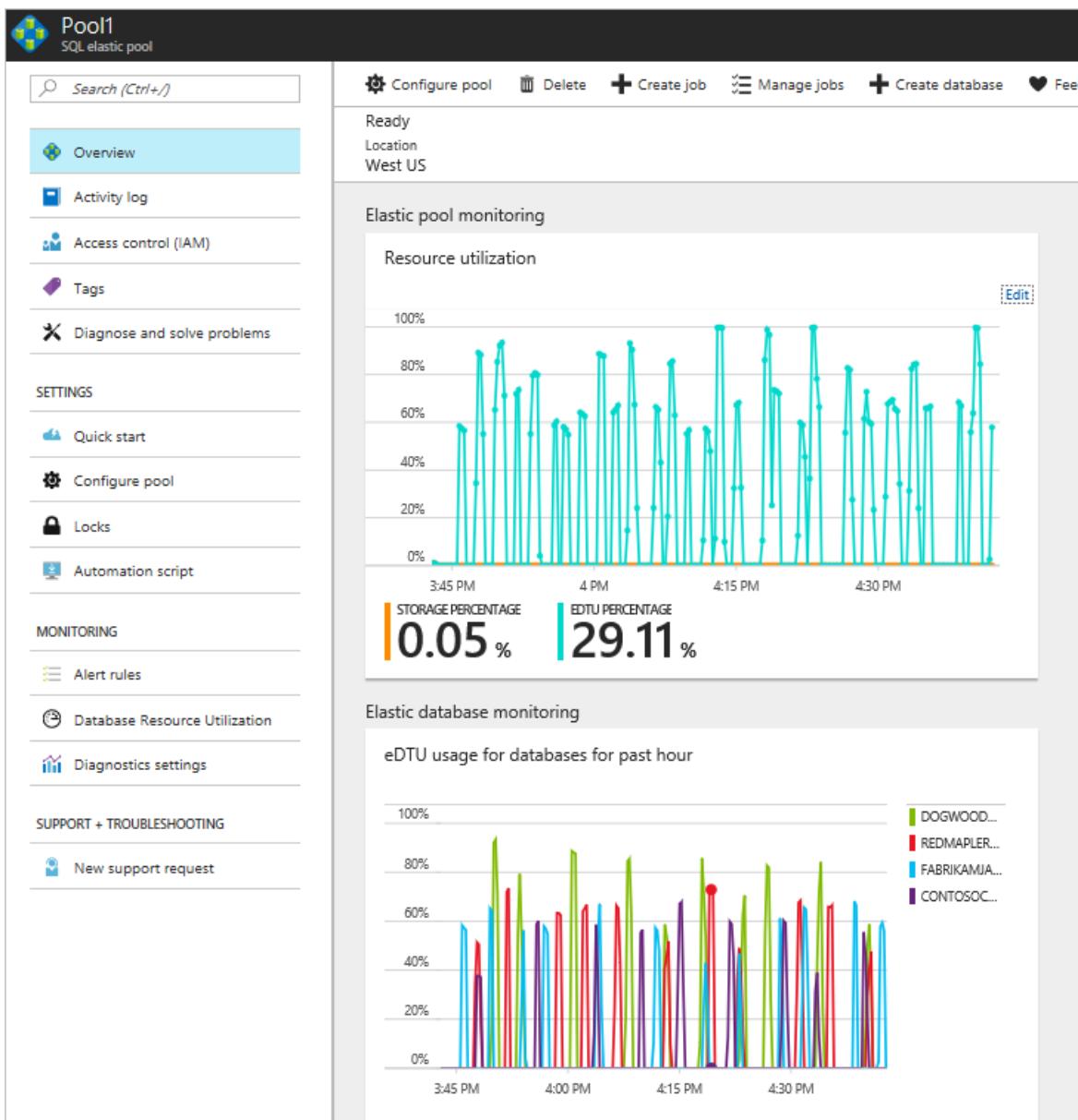
The screenshot shows the Microsoft Azure portal interface for managing SQL databases. The left sidebar lists various management options like Overview, Activity log, and Diagnose and solve problems. The main content area displays the 'Databases' section under 'SQL databases', showing four databases: contosoconcerthall, dogwooddojo, fabrikamjazzclub, and redmapleracing, all in an Online status and Elastic Standard tier. Below this is the 'Elastic database pools' section, which shows one pool named 'Pool1' with a Standard Pool tier and 50 EDTU. Both the database table and the pool table are highlighted with red boxes.

NAME	PRICING TIER	POOL EDTU
Pool1	Standard Pool	50

## Monitor the pool

After `LoadGenerator.ps1` runs for several minutes, enough data should be available to start looking at some monitoring capabilities. These capabilities are built into pools and databases.

Browse to the server **tenants1-dpt-<user>**, and select **Pool1** to view resource utilization for the pool. In the following charts, the load generator ran for one hour.



- The first chart, labeled **Resource utilization**, shows pool eDTU utilization.
- The second chart shows eDTU utilization of the five most active databases in the pool.

The two charts illustrate that elastic pools and SQL Database are well suited to unpredictable SaaS application workloads. The charts show that four databases are each bursting to as much as 40 eDTUs, and yet all the databases are comfortably supported by a 50-eDTU pool. The 50-eDTU pool can support even heavier workloads. If the databases are provisioned as single databases, each one needs to be an S2 (50 DTU) to support the bursts. The cost of four single S2 databases is nearly three times the price of the pool. In real-world situations, SQL Database customers run up to 500 databases in 200 eDTU pools. For more information, see the [Performance monitoring tutorial](#).

## Additional resources

- For more information, see additional [tutorials that build on the Wingtip Tickets SaaS database-per-tenant application](#).
- To learn about elastic pools, see [What is an Azure SQL elastic pool?](#).
- To learn about elastic jobs, see [Manage scaled-out cloud databases](#).
- To learn about multitenant SaaS applications, see [Design patterns for multitenant SaaS applications](#).

## Next steps

In this tutorial you learned:

- How to deploy the Wingtip Tickets SaaS application.
- About the servers, pools, and databases that make up the app.
- How tenants are mapped to their data with the *catalog*.
- How to provision new tenants.
- How to view pool utilization to monitor tenant activity.
- How to delete sample resources to stop related billing.

Next, try the [Provision and catalog tutorial](#).

# Learn how to provision new tenants and register them in the catalog

11/15/2019 • 9 minutes to read • [Edit Online](#)

In this tutorial, you learn how to provision and catalog SaaS patterns. You also learn how they're implemented in the Wingtip Tickets SaaS database-per-tenant application. You create and initialize new tenant databases and register them in the application's tenant catalog. The catalog is a database that maintains the mapping between the SaaS application's many tenants and their data. The catalog plays an important role in directing application and management requests to the correct database.

In this tutorial, you learn how to:

- Provision a single new tenant.
- Provision a batch of additional tenants.

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS database-per-tenant app is deployed. To deploy it in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS database-per-tenant application](#).
- Azure PowerShell is installed. For more information, see [Get started with Azure PowerShell](#).

## Introduction to the SaaS catalog pattern

In a database-backed multitenant SaaS application, it's important to know where information for each tenant is stored. In the SaaS catalog pattern, a catalog database is used to hold the mapping between each tenant and the database in which their data is stored. This pattern applies whenever tenant data is distributed across multiple databases.

Each tenant is identified by a key in the catalog, which is mapped to the location of their database. In the Wingtip Tickets app, the key is formed from a hash of the tenant's name. This scheme allows the app to construct the key from the tenant name included in the application URL. Other tenant key schemes can be used.

The catalog allows the name or location of the database to be changed with minimal impact on the application. In a multitenant database model, this capability also accommodates moving a tenant between databases. The catalog also can be used to indicate whether a tenant or database is offline for maintenance or other actions. This capability is explored in the [Restore single tenant tutorial](#).

The catalog also can store additional tenant or database metadata, such as the schema version, service plan, or SLAs offered to tenants. The catalog can store other information that enables application management, customer support, or DevOps.

Beyond the SaaS application, the catalog can enable database tools. In the Wingtip Tickets SaaS database-per-tenant sample, the catalog is used to enable cross-tenant query, which is explored in the [Ad hoc reporting tutorial](#). Cross-database job management is explored in the [Schema management](#) and [Tenant analytics](#) tutorials.

In the Wingtip Tickets SaaS samples, the catalog is implemented by using the Shard Management features of the [Elastic Database client library \(EDCL\)](#). The EDCL is available in Java and the .NET Framework. The EDCL enables an application to create, manage, and use a database-backed shard map.

A shard map contains a list of shards (databases) and the mapping between keys (tenants) and shards. EDCL functions are used during tenant provisioning to create the entries in the shard map. They're used at run time by applications to connect to the correct database. EDCL caches connection information to minimize traffic to the

catalog database and speed up the application.

#### IMPORTANT

The mapping data is accessible in the catalog database, but *don't edit it*. Edit mapping data by using Elastic Database Client Library APIs only. Directly manipulating the mapping data risks corrupting the catalog and isn't supported.

## Introduction to the SaaS provisioning pattern

When you add a new tenant in a SaaS application that uses a single-tenant database model, you must provision a new tenant database. The database must be created in the appropriate location and service tier. It also must be initialized with the appropriate schema and reference data. And it must be registered in the catalog under the appropriate tenant key.

Different approaches to database provisioning can be used. You can execute SQL scripts, deploy a bacpac, or copy a template database.

Database provisioning needs to be part of your schema management strategy. You must make sure that new databases are provisioned with the latest schema. This requirement is explored in the [Schema management tutorial](#).

The Wingtip Tickets database-per-tenant app provisions new tenants by copying a template database named *basetenantdb*, which is deployed on the catalog server. Provisioning can be integrated into the application as part of a sign-up experience. It also can be supported offline by using scripts. This tutorial explores provisioning by using PowerShell.

Provisioning scripts copy the *basetenantdb* database to create a new tenant database in an elastic pool. The tenant database is created in the tenant server mapped to the *newtenant* DNS alias. This alias maintains a reference to the server used to provision new tenants and is updated to point to a recovery tenant server in the disaster recovery tutorials ([DR using georestore](#), [DR using georeplication](#)). The scripts then initialize the database with tenant-specific information and register it in the catalog shard map. Tenant databases are given names based on the tenant name. This naming scheme isn't a critical part of the pattern. The catalog maps the tenant key to the database name, so any naming convention can be used.

## Get the Wingtip Tickets SaaS database-per-tenant application scripts

The Wingtip Tickets SaaS scripts and application source code are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

## Provision and catalog detailed walkthrough

To understand how the Wingtip Tickets application implements new tenant provisioning, add a breakpoint and follow the workflow while you provision a tenant.

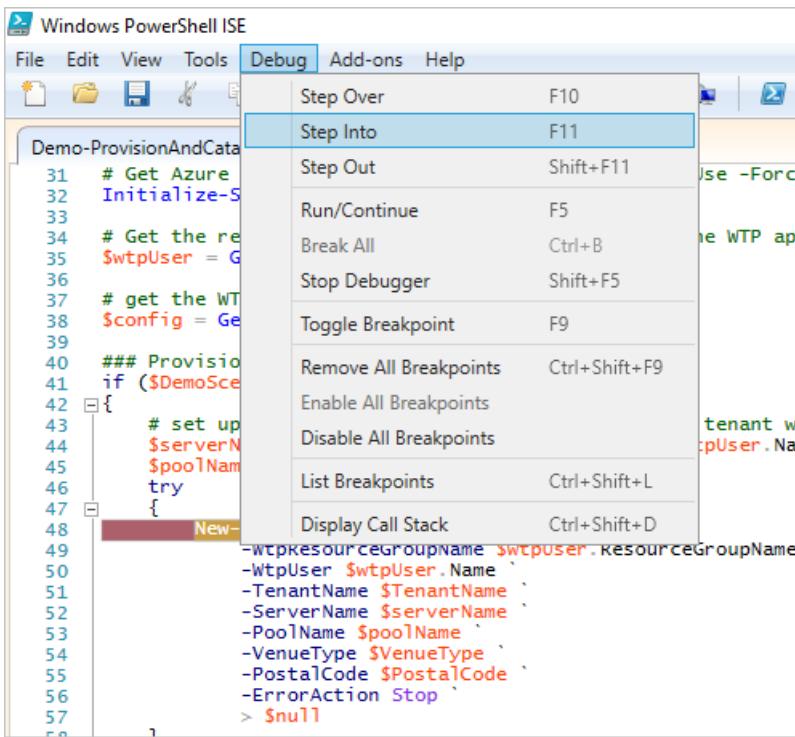
1. In the PowerShell ISE, open ...\\Learning Modules\\ProvisionAndCatalog\\Demo-ProvisionAndCatalog.ps1 and set the following parameters:
  - **\$TenantName** = the name of the new venue (for example, *Bushwillow Blues*).
  - **\$VenueType** = one of the predefined venue types: *blues, classicalmusic, dance, jazz, judo, motor racing, multipurpose, opera, rockmusic, soccer*.
  - **\$DemoScenario** = 1, *Provision a single tenant*.
2. To add a breakpoint, put your cursor anywhere on the line that says *New-Tenant* `. Then press F9.

```

40 ### Provision a single tenant
41 if ($DemoScenario -eq 1)
42 {
43 # set up the server and pool names in which the tenant will be provisioned
44 $serverName = $config.TenantServerNameStem + $wtpUser.Name
45 $poolName = $config.TenantPoolNameStem + "1"
46 try
47 {
48 New-Tenant
49 -WtpResourceGroupName $wtpUser.ResourceGroupName
50 -WtpUser $wtpUser.Name
51 -TenantName $tenantName
52 -ServerName $serverName
53 -PoolName $poolName
54 -VenueType $venueType
55 -PostalCode $postalCode
56 -ErrorAction Stop
57 > $null
58 }
59 catch
60 {
61 Write-Error $_.Exception.Message
62 exit
63 }

```

3. To run the script, press F5.
4. After the script execution stops at the breakpoint, press F11 to step into the code.



Trace the script's execution by using the **Debug** menu options. Press F10 and F11 to step over or into the called functions. For more information about debugging PowerShell scripts, see [Tips on working with and debugging PowerShell scripts](#).

You don't need to explicitly follow this workflow. It explains how to debug the script.

- **Import the CatalogAndDatabaseManagement.psm1 module.** It provides a catalog and tenant-level abstraction over the [Shard Management](#) functions. This module encapsulates much of the catalog pattern and is worth exploring.
- **Import the SubscriptionManagement.psm1 module.** It contains functions for signing in to Azure and selecting the Azure subscription you want to work with.
- **Get configuration details.** Step into Get-Configuration by using F11, and see how the app config is specified. Resource names and other app-specific values are defined here. Don't change these values until you are familiar with the scripts.
- **Get the catalog object.** Step into Get-Catalog, which composes and returns a catalog object that's used in

the higher-level script. This function uses Shard Management functions that are imported from **AzureShardManagement.psm1**. The catalog object is composed of the following elements:

- \$catalogServerFullyQualifiedName is constructed by using the standard stem plus your user name:  
*catalog-<user>.database.windows .net*.
- \$catalogDatabaseName is retrieved from the config: *tenantcatalog*.
- \$shardMapManager object is initialized from the catalog database.
- \$shardMap object is initialized from the *tenantcatalog* shard map in the catalog database. A catalog object is composed and returned. It's used in the higher-level script.
- **Calculate the new tenant key.** A hash function is used to create the tenant key from the tenant name.
- **Check if the tenant key already exists.** The catalog is checked to make sure the key is available.
- **The tenant database is provisioned with New-TenantDatabase.** Use F11 to step into how the database is provisioned by using an [Azure Resource Manager template](#).

The database name is constructed from the tenant name to make it clear which shard belongs to which tenant. You also can use other database naming conventions. A Resource Manager template creates a tenant database by copying a template database (*baseTenantDB*) on the catalog server. As an alternative, you can create a database and initialize it by importing a bacpac. Or you can execute an initialization script from a well-known location.

The Resource Manager template is in the ...\\Learning Modules\\Common\\ folder:  
*tenantdatabasecopytemplate.json*

- **The tenant database is further initialized.** The venue (tenant) name and the venue type are added. You also can do other initialization here.
- **The tenant database is registered in the catalog.** It's registered with *Add-TenantDatabaseToCatalog* by using the tenant key. Use F11 to step into the details:
  - The catalog database is added to the shard map (the list of known databases).
  - The mapping that links the key value to the shard is created.
  - Additional metadata about the tenant (the venue's name) is added to the Tenants table in the catalog. The Tenants table isn't part of the Shard Management schema, and it isn't installed by the EDCL. This table illustrates how the catalog database can be extended to support additional application-specific data.

After provisioning completes, execution returns to the original *Demo-ProvisionAndCatalog* script. The **Events** page opens for the new tenant in the browser.

ALL SESSIONS

WELCOME [SIGN IN]

Next | Event 3

Tickets >

MAY 8 MON Event 3 Performer 3 Tickets

MAY 11 THU Event 4 Performer 4 Tickets

mytickets  
Update your list of favorites and never miss an event!  
Sign In

## Provision a batch of tenants

This exercise provisions a batch of 17 tenants. We recommend that you provision this batch of tenants before starting other Wingtip Tickets SaaS database-per-tenant tutorials. There are more than just a few databases to work with.

1. In the PowerShell ISE, open ...\\Learning Modules\\ProvisionAndCatalog\\*Demo-ProvisionAndCatalog.ps1*. Change the `$DemoScenario` parameter to 3:
  - **\$DemoScenario = 3, Provision a batch of tenants.**
2. To run the script, press F5.

The script deploys a batch of additional tenants. It uses an [Azure Resource Manager template](#) that controls the batch and delegates provisioning of each database to a linked template. Using templates in this way allows Azure Resource Manager to broker the provisioning process for your script. The templates provision databases in parallel and handle retries, if needed. The script is idempotent, so if it fails or stops for any reason, run it again.

### Verify the batch of tenants that successfully deployed

- In the [Azure portal](#), browse to your list of servers and open the *tenants1* server. Select **SQL databases**, and verify that the batch of 17 additional databases is now in the list.

**Resource group (change)**

**Status**  
Available

**Location**  
West US 2

**Subscription name (change)**  
**SQL DB Content**

**Subscription ID**

**Databases**

**SQL databases**

**21 Databases**

<input type="checkbox"/> DATABASE	STATUS	PRICING TIER
papayaplayers	Online	Elastic Standard
cottonwoodconcerthall	Online	Elastic Standard
dogwooddojo	Online	Elastic Standard
osageopera	Online	Elastic Standard
blueoakjazzclub	Online	Elastic Standard
limetreetrack	Online	Elastic Standard
juniperjammersjazz	Online	Elastic Standard

[See more](#)

## Other provisioning patterns

Other provisioning patterns not included in this tutorial:

**Pre-provisioning databases:** The pre-provisioning pattern exploits the fact that databases in an elastic pool don't add extra cost. Billing is for the elastic pool, not the databases. Idle databases consume no resources. By pre-provisioning databases in a pool and allocating them when needed, you can reduce the time to add tenants. The number of databases pre-provisioned can be adjusted as needed to keep a buffer suitable for the anticipated provisioning rate.

**Auto-provisioning:** In the auto-provisioning pattern, a provisioning service provisions servers, pools, and databases automatically, as needed. If you want, you can include pre-provisioning databases in elastic pools. If databases are decommissioned and deleted, gaps in elastic pools can be filled by the provisioning service. Such a service can be simple or complex, such as handling provisioning across multiple geographies and setting up geo-replication for disaster recovery.

With the auto-provisioning pattern, a client application or script submits a provisioning request to a queue to be processed by the provisioning service. It then polls the service to determine completion. If pre-provisioning is used, requests are handled quickly. The service provisions a replacement database in the background.

## Next steps

In this tutorial you learned how to:

- Provision a single new tenant.
- Provision a batch of additional tenants.
- Step into the details of provisioning tenants and registering them into the catalog.

Try the [Performance monitoring tutorial](#).

## Additional resources

- Additional [tutorials that build on the Wingtip Tickets SaaS database-per-tenant application](#)
- [Elastic database client library](#)
- [Debug scripts in the Windows PowerShell ISE](#)

# Monitor and manage performance of Azure SQL databases and pools in a multi-tenant SaaS app

11/7/2019 • 15 minutes to read • [Edit Online](#)

In this tutorial, several key performance management scenarios used in SaaS applications are explored. Using a load generator to simulate activity across all tenant databases, the built-in monitoring and alerting features of SQL Database and elastic pools are demonstrated.

The Wingtip Tickets SaaS Database Per Tenant app uses a single-tenant data model, where each venue (tenant) has their own database. Like many SaaS applications, the anticipated tenant workload pattern is unpredictable and sporadic. In other words, ticket sales may occur at any time. To take advantage of this typical database usage pattern, tenant databases are deployed into elastic pools. Elastic pools optimize the cost of a solution by sharing resources across many databases. With this type of pattern, it's important to monitor database and pool resource usage to ensure that loads are reasonably balanced across pools. You also need to ensure that individual databases have adequate resources, and that pools are not hitting their [eDTU](#) limits. This tutorial explores ways to monitor and manage databases and pools, and how to take corrective action in response to variations in workload.

In this tutorial you learn how to:

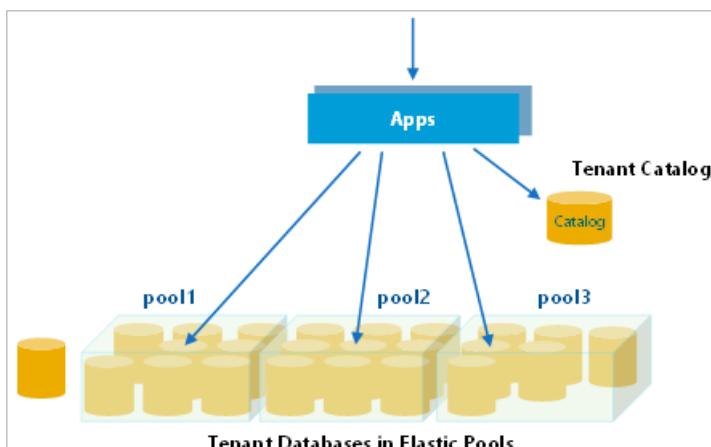
- Simulate usage on the tenant databases by running a provided load generator
- Monitor the tenant databases as they respond to the increase in load
- Scale up the Elastic pool in response to the increased database load
- Provision a second Elastic pool to load balance database activity

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS Database Per Tenant app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Database Per Tenant application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)

## Introduction to SaaS performance management patterns

Managing database performance consists of compiling and analyzing performance data, and then reacting to this data by adjusting parameters to maintain an acceptable response time for your application. When hosting multiple tenants, Elastic pools are a cost-effective way to provide and manage resources for a group of databases with unpredictable workloads. With certain workload patterns, as few as two S3 databases can benefit from being managed in a pool.



Pools, and the databases in pools, should be monitored to ensure they stay within acceptable ranges of performance. Tune the pool configuration to meet the needs of the aggregate workload of all databases, ensuring that the pool eDTUs are appropriate for the overall workload. Adjust the per-database min and per-database max eDTU values to appropriate values for your specific application requirements.

### Performance management strategies

- To avoid having to manually monitor performance, it's most effective to **set alerts that trigger when databases or pools stray out of normal ranges**.
- To respond to short-term fluctuations in the aggregate compute size of a pool, the **pool eDTU level can be scaled up or down**. If this fluctuation occurs on a regular or predictable basis, **scaling the pool can be scheduled to occur automatically**. For example, scale down when you know your workload is light, maybe overnight, or during weekends.
- To respond to longer-term fluctuations, or changes in the number of databases, **individual databases can be moved into other pools**.
- To respond to short-term increases in *individual* database load **individual databases can be taken out of a pool and assigned an individual compute size**. Once the load is reduced, the database can then be returned to the pool. When this is known in advance, databases can be moved pre-emptively to ensure the database always has the resources it needs, and to avoid impact on other databases in the pool. If this requirement is predictable, such as a venue experiencing a rush of ticket sales for a popular event, then this management behavior can be integrated into the application.

The [Azure portal](#) provides built-in monitoring and alerting on most resources. For SQL Database, monitoring and alerting is available on databases and pools. This built-in monitoring and alerting is resource-specific, so it's convenient to use for small numbers of resources, but is not very convenient when working with many resources.

For high-volume scenarios, where you're working with many resources, [Azure Monitor logs](#) can be used. This is a separate Azure service that provides analytics over emitted diagnostic logs and telemetry gathered in a Log Analytics workspace. Azure Monitor logs can collect telemetry from many services and be used to query and set alerts.

## Get the Wingtip Tickets SaaS Database Per Tenant application scripts

The Wingtip Tickets SaaS Multi-tenant Database scripts and application source code are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

## Provision additional tenants

While pools can be cost-effective with just two S3 databases, the more databases that are in the pool the more cost-effective the averaging effect becomes. For a good understanding of how performance monitoring and management works at scale, this tutorial requires you have at least 20 databases deployed.

If you already provisioned a batch of tenants in a prior tutorial, skip to the [Simulate usage on all tenant databases](#) section.

1. In the **PowerShell ISE**, open ...\\Learning Modules\\Performance Monitoring and Management\\*Demo-PerformanceMonitoringAndManagement.ps1*. Keep this script open as you'll run several scenarios during this tutorial.
2. Set **\$DemoScenario = 1, Provision a batch of tenants**
3. Press **F5** to run the script.

The script will deploy 17 tenants in less than five minutes.

The *New-TenantBatch* script uses a nested or linked set of [Resource Manager](#) templates that create a batch of

tenants, which by default copies the database **basetenantdb** on the catalog server to create the new tenant databases, then registers these in the catalog, and finally initializes them with the tenant name and venue type. This is consistent with the way the app provisions a new tenant. Any changes made to *basetenantdb* are applied to any new tenants provisioned thereafter. See the [Schema Management tutorial](#) to see how to make schema changes to *existing* tenant databases (including the *basetenantdb* database).

## Simulate usage on all tenant databases

The *Demo-PerformanceMonitoringAndManagement.ps1* script is provided that simulates a workload running against all tenant databases. The load is generated using one of the available load scenarios:

DEMO	SCENARIO
2	Generate normal intensity load (approximately 40 DTU)
3	Generate load with longer and more frequent bursts per database
4	Generate load with higher DTU bursts per database (approximately 80 DTU)
5	Generate a normal load plus a high load on a single tenant (approximately 95 DTU)
6	Generate unbalanced load across multiple pools

The load generator applies a *synthetic* CPU-only load to every tenant database. The generator starts a job for each tenant database, which calls a stored procedure periodically that generates the load. The load levels (in eDTUs), duration, and intervals are varied across all databases, simulating unpredictable tenant activity.

1. In the **PowerShell ISE**, open ...\\Learning Modules\\Performance Monitoring and Management\\*Demo-PerformanceMonitoringAndManagement.ps1*. Keep this script open as you'll run several scenarios during this tutorial.
2. Set **\$DemoScenario = 2, Generate normal intensity load**.
3. Press **F5** to apply a load to all your tenant databases.

Wingtip Tickets SaaS Database Per Tenant is a SaaS app, and the real-world load on a SaaS app is typically sporadic and unpredictable. To simulate this, the load generator produces a randomized load distributed across all tenants. Several minutes are needed for the load pattern to emerge, so run the load generator for 3-5 minutes before attempting to monitor the load in the following sections.

### IMPORTANT

The load generator is running as a series of jobs in your local PowerShell session. Keep the *Demo-PerformanceMonitoringAndManagement.ps1* tab open! If you close the tab, or suspend your machine, the load generator stops. The load generator remains in a *job-invoking* state where it generates load on any new tenants that are provisioned after the generator is started. Use *Ctrl-C* to stop invoking new jobs and exit the script. The load generator will continue to run, but only on existing tenants.

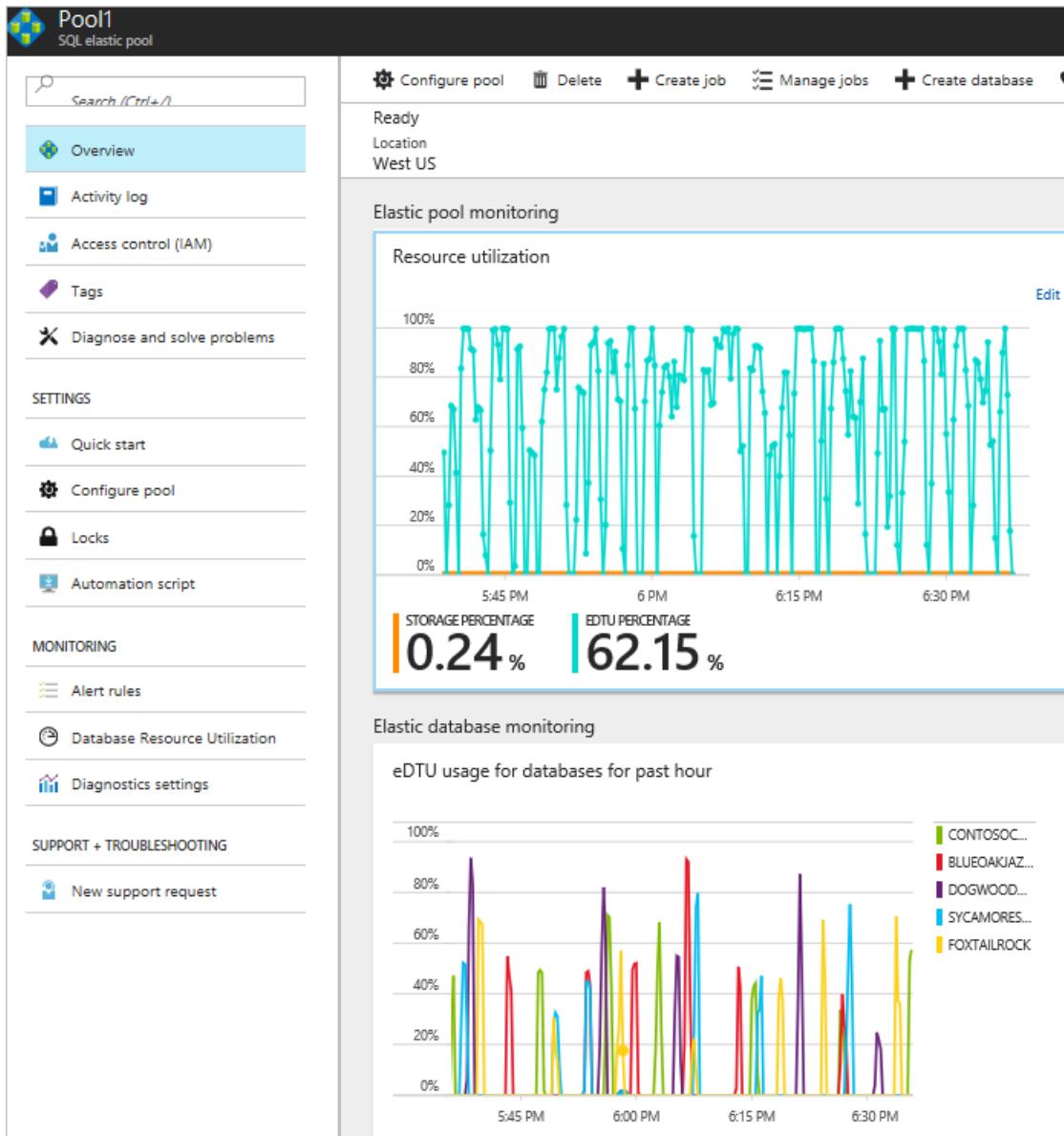
## Monitor resource usage using the Azure portal

To monitor the resource usage that results from the load being applied, open the portal to the pool containing the tenant databases:

1. Open the [Azure portal](#) and browse to the *tenants1-dpt-<USER>* server.
2. Scroll down and locate elastic pools and click **Pool1**. This pool contains all the tenant databases created so far.

Observe the **Elastic pool monitoring** and **Elastic database monitoring** charts.

The pool's resource utilization is the aggregate database utilization for all databases in the pool. The database chart shows the five hottest databases:



Because there are additional databases in the pool beyond the top five, the pool utilization shows activity that is not reflected in the top five databases chart. For additional details, click **Database Resource Utilization**:

 Pool1 - Database Resource Utilization

SQL elastic pool

🔍 Search (Ctrl+I)

- 🌐 Overview
- 📅 Activity log
- 👤 Access control (IAM)
- 🏷️ Tags
- ✖️ Diagnose and solve problems

**SETTINGS**

- 💨 Quick start
- ⚙️ Configure pool
- 🔒 Locks
- 💻 Automation script

**MONITORING**

- 📊 Alert rules
- ⌚ Database Resource Utilization
- 📈 Diagnostics settings

📝 Edit Chart    ❤️ Feedback

Select additional metrics to display below:

0 selected
Total: 6
▼

Elastic databases within the pool (select up to five)

🔍 Search to filter databases...

DATABASE NAME	^	AVG EDTU	PEAK EDTU
<span style="font-size: 1.5em;">📁</span> hornbeamhiphop	4.32	93.65	
<span style="font-size: 1.5em;">📁</span> blueoakjazzclub	3.848	88.26	
<span style="font-size: 1.5em;">📁</span> papayaplayers	3.636	75.28	
<span style="font-size: 1.5em;">📁</span> juniperjammersjazz	3.321	85.89	
<span style="font-size: 1.5em;">📁</span> fabrikamjazzclub	3.198	84.09	
<span style="font-size: 1.5em;">📁</span> redmapleracing	3.091	62.4	
<span style="font-size: 1.5em;">📁</span> mahoganysoccer	3.036	65.61	
<span style="font-size: 1.5em;">📁</span> magnoliamotorracing	2.911	70.3	
<span style="font-size: 1.5em;">📁</span> contosoconcerthall	2.889	71.51	
<span style="font-size: 1.5em;">📁</span> dogwooddojo	2.731	57.49	
<span style="font-size: 1.5em;">📁</span> osageopera	2.647	75.15	

## Set performance alerts on the pool

Set an alert on the pool that triggers on >75% utilization as follows:

1. Open *Pool1* (on the *tenants1-dpt-<user>* server) in the [Azure portal](#).
2. Click **Alert Rules**, and then click **+ Add alert**:

Microsoft Azure Pool1 - Alert rules

Pool1 - Alert rules  
SQL elastic pool

+ Add alert

NAME

You haven't created any alert rules.

OVERVIEW

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Quick start

Configure pool

Locks

Automation script

MONITORING

Alert rules

Database Resource Utilization

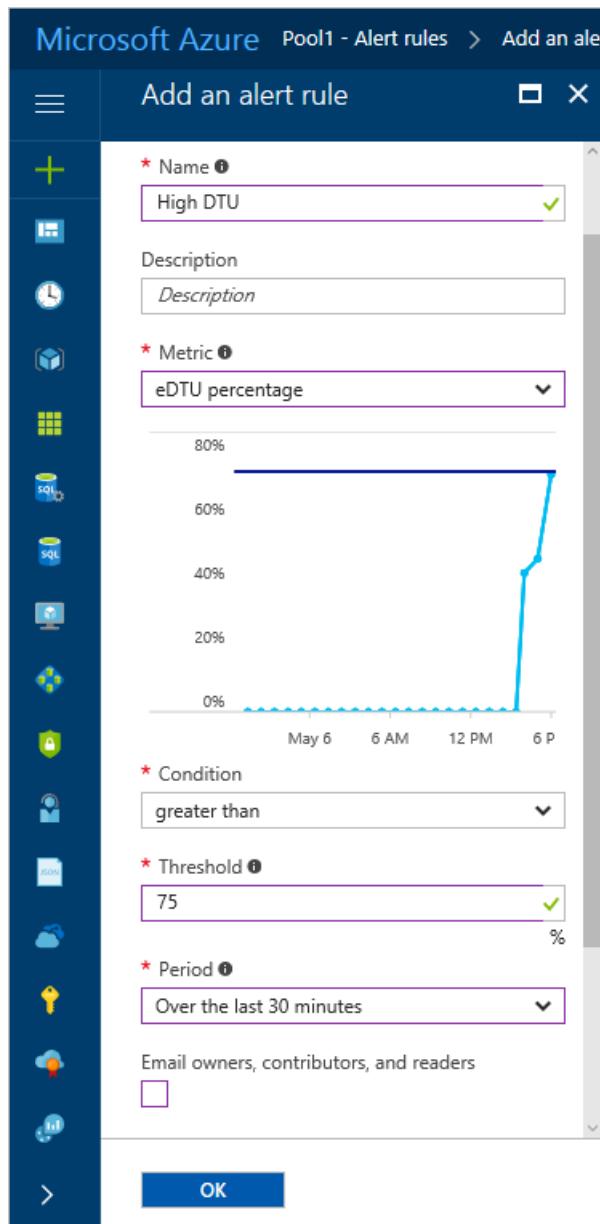
Diagnostics settings

3. Provide a name, such as **High DTU**,

4. Set the following values:

- **Metric = eDTU percentage**
- **Condition = greater than**
- **Threshold = 75**
- **Period = Over the last 30 minutes**

5. Add an email address to the *Additional administrator email(s)* box and click **OK**.



## Scale up a busy pool

If the aggregate load level increases on a pool to the point that it maxes out the pool and reaches 100% eDTU usage, then individual database performance is affected, potentially slowing query response times for all databases in the pool.

**Short-term**, consider scaling up the pool to provide additional resources, or removing databases from the pool (moving them to other pools, or out of the pool to a stand-alone service tier).

**Longer term**, consider optimizing queries or index usage to improve database performance. Depending on the application's sensitivity to performance issues its best practice to scale a pool up before it reaches 100% eDTU usage. Use an alert to warn you in advance.

You can simulate a busy pool by increasing the load produced by the generator. Causing the databases to burst more frequently, and for longer, increasing the aggregate load on the pool without changing the requirements of the individual databases. Scaling up the pool is easily done in the portal or from PowerShell. This exercise uses the portal.

1. Set \$DemoScenario = 3, *Generate load with longer and more frequent bursts per database* to increase the intensity of the aggregate load on the pool without changing the peak load required by each database.
2. Press **F5** to apply a load to all your tenant databases.

### 3. Go to **Pool1** in the Azure portal.

Monitor the increased pool eDTU usage on the upper chart. It takes a few minutes for the new higher load to kick in, but you should quickly see the pool start to hit max utilization, and as the load steadies into the new pattern, it rapidly overloads the pool.

1. To scale up the pool, click **Configure pool** at the top of the **Pool1** page.
2. Adjust the **Pool eDTU** setting to **100**. Changing the pool eDTU does not change the per-database settings (which is still 50 eDTU max per database). You can see the per-database settings on the right side of the **Configure pool** page.
3. Click **Save** to submit the request to scale the pool.

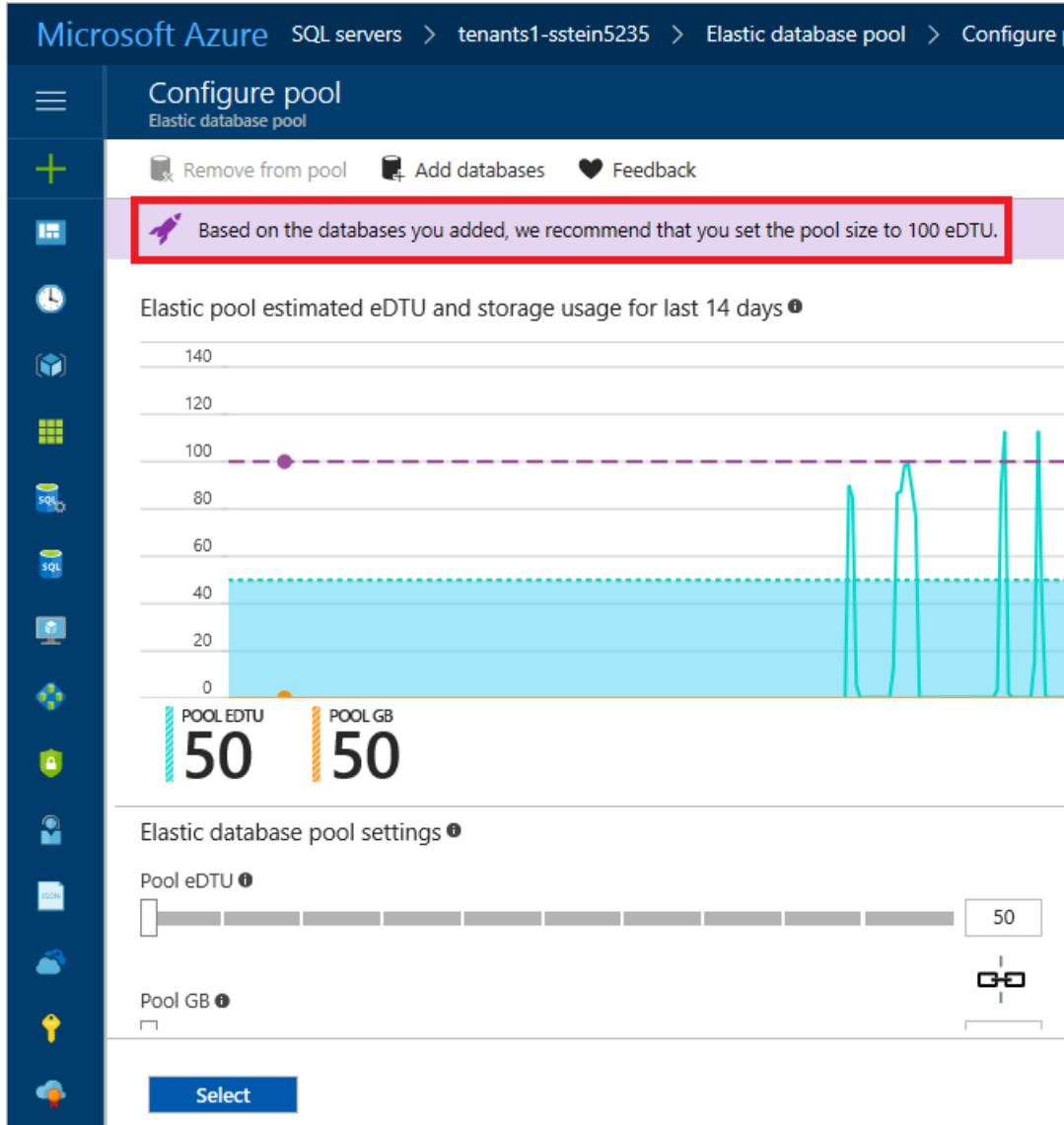
Go back to **Pool1 > Overview** to view the monitoring charts. Monitor the effect of providing the pool with more resources (although with few databases and a randomized load it's not always easy to see conclusively until you run for some time). While you are looking at the charts bear in mind that 100% on the upper chart now represents 100 eDTUs, while on the lower chart 100% is still 50 eDTUs as the per-database max is still 50 eDTUs.

Databases remain online and fully available throughout the process. At the last moment as each database is ready to be enabled with the new pool eDTU, any active connections are broken. Application code should always be written to retry dropped connections, and so will reconnect to the database in the scaled-up pool.

## Load balance between pools

As an alternative to scaling up the pool, create a second pool and move databases into it to balance the load between the two pools. To do this the new pool must be created on the same server as the first.

1. In the [Azure portal](#), open the **tenants1-dpt-<USER>** server.
2. Click **+ New pool** to create a pool on the current server.
3. On the **Elastic pool** template:
  - a. Set **Name** to *Pool2*.
  - b. Leave the pricing tier as **Standard Pool**.
  - c. Click **Configure pool**,
  - d. Set **Pool eDTU** to **50 eDTU**.
  - e. Click **Add databases** to see a list of databases on the server that can be added to *Pool2*.
  - f. Select any 10 databases to move these to the new pool, and then click **Select**. If you've been running the load generator, the service already knows that your performance profile requires a larger pool than the default 50 eDTU size and recommends starting with a 100 eDTU setting.



g. For this tutorial, leave the default at 50 eDTUs, and click **Select** again.

h. Select **OK** to create the new pool and to move the selected databases into it.

Creating the pool and moving the databases takes a few minutes. As databases are moved they remain online and fully accessible until the very last moment, at which point any open connections are closed. As long as you have some retry logic, clients will then connect to the database in the new pool.

Browse to **Pool2** (on the *tenants1-dpt-<user>* server) to open the pool and monitor its performance. If you don't see it, wait for provisioning of the new pool to complete.

You now see that resource usage on *Pool1* has dropped and that *Pool2* is now similarly loaded.

## Manage performance of an individual database

If an individual database in a pool experiences a sustained high load, depending on the pool configuration, it may tend to dominate the resources in the pool and impact other databases. If the activity is likely to continue for some time, the database can be temporarily moved out of the pool. This allows the database to have the extra resources it needs, and isolates it from the other databases.

This exercise simulates the effect of Contoso Concert Hall experiencing a high load when tickets go on sale for a popular concert.

1. In the **PowerShell ISE**, open the ...\\Demo-PerformanceMonitoringAndManagement.ps1 script.
2. Set **\$DemoScenario = 5, Generate a normal load plus a high load on a single tenant**

(approximately 95 DTU).

3. Set **\$SingleTenantDatabaseName = contosoconcerthall**
4. Execute the script using **F5**.
5. In the [Azure portal](#), browse to the list of databases on the *tenants1-dpt-<user>* server.
6. Click on the **contosoconcerthall** database.
7. Click on the pool that **contosoconcerthall** is in. Locate the pool in the **Elastic pool** section.
8. Inspect the **Elastic pool monitoring** chart and look for the increased pool eDTU usage. After a minute or two, the higher load should start to kick in, and you should quickly see that the pool hits 100% utilization.
9. Inspect the **Elastic database monitoring** display, which shows the hottest databases in the past hour. The *contosoconcerthall* database should soon appear as one of the five hottest databases.
10. **Click on the Elastic database monitoring chart** and it opens the **Database Resource Utilization** page where you can monitor any of the databases. This lets you isolate the display for the *contosoconcerthall* database.
11. From the list of databases, click **contosoconcerthall**.
12. Click **Pricing Tier (scale DTUs)** to open the **Configure performance** page where you can set a stand-alone compute size for the database.
13. Click on the **Standard** tab to open the scale options in the Standard tier.
14. Slide the **DTU slider** to right to select **100** DTUs. Note this corresponds to the service objective, **S3**.
15. Click **Apply** to move the database out of the pool and make it a *Standard S3* database.
16. Once scaling is complete, monitor the effect on the *contosoconcerthall* database and Pool1 on the elastic pool and database blades.

Once the high load on the *contosoconcerthall* database subsides you should promptly return it to the pool to reduce its cost. If it's unclear when that will happen you could set an alert on the database that will trigger when its DTU usage drops below the per-database max on the pool. Moving a database into a pool is described in exercise 5.

## Other performance management patterns

**Pre-emptive scaling** In the exercise above where you explored how to scale an isolated database, you knew which database to look for. If the management of Contoso Concert Hall had informed Wingtips of the impending ticket sale, the database could have been moved out of the pool pre-emptively. Otherwise, it would likely have required an alert on the pool or the database to spot what was happening. You wouldn't want to learn about this from the other tenants in the pool complaining of degraded performance. And if the tenant can predict how long they will need additional resources you can set up an Azure Automation runbook to move the database out of the pool and then back in again on a defined schedule.

**Tenant self-service scaling** Because scaling is a task easily called via the management API, you can easily build the ability to scale tenant databases into your tenant-facing application, and offer it as a feature of your SaaS service. For example, let tenants self-administer scaling up and down, perhaps linked directly to their billing!

### Scaling a pool up and down on a schedule to match usage patterns

Where aggregate tenant usage follows predictable usage patterns, you can use Azure Automation to scale a pool up and down on a schedule. For example, scale a pool down after 6pm and up again before 6am on weekdays when you know there is a drop in resource requirements.

## Next steps

In this tutorial you learn how to:

- Simulate usage on the tenant databases by running a provided load generator
- Monitor the tenant databases as they respond to the increase in load
- Scale up the Elastic pool in response to the increased database load
- Provision a second Elastic pool to load balance the database activity

[Restore a single tenant tutorial](#)

## Additional resources

- Additional [tutorials that build upon the Wingtip Tickets SaaS Database Per Tenant application deployment](#)
- [SQL Elastic pools](#)
- [Azure automation](#)
- [Azure Monitor logs](#) - Setting up and using Azure Monitor logs tutorial

# Set up and use Azure Monitor logs with a multitenant SQL Database SaaS app

11/7/2019 • 5 minutes to read • [Edit Online](#)

In this tutorial, you set up and use [Azure Monitor logs](#) to monitor elastic pools and databases. This tutorial builds on the [Performance monitoring and management tutorial](#). It shows how to use Azure Monitor logs to augment the monitoring and alerting provided in the Azure portal. Azure Monitor logs supports monitoring thousands of elastic pools and hundreds of thousands of databases. Azure Monitor logs provides a single monitoring solution, which can integrate monitoring of different applications and Azure services across multiple Azure subscriptions.

## NOTE

This article was recently updated to use the term Azure Monitor logs instead of Log Analytics. Log data is still stored in a Log Analytics workspace and is still collected and analyzed by the same Log Analytics service. We are updating the terminology to better reflect the role of logs in Azure Monitor. See [Azure Monitor terminology changes](#) for details.

In this tutorial you learn how to:

- Install and configure Azure Monitor logs.
- Use Azure Monitor logs to monitor pools and databases.

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS database-per-tenant app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS database-per-tenant application](#).
- Azure PowerShell is installed. For more information, see [Get started with Azure PowerShell](#).

See the [Performance monitoring and management tutorial](#) for a discussion of SaaS scenarios and patterns and how they affect the requirements on a monitoring solution.

## Monitor and manage database and elastic pool performance with Azure Monitor logs

For Azure SQL Database, monitoring and alerting is available on databases and pools in the Azure portal. This built-in monitoring and alerting is convenient, but it's also resource-specific. That means it's less well suited to monitor large installations or provide a unified view across resources and subscriptions.

For high-volume scenarios, you can use Azure Monitor logs for monitoring and alerting. Azure Monitor is a separate Azure service that enables analytics over diagnostic logs and telemetry that's gathered in a workspace from potentially many services. Azure Monitor logs provides a built-in query language and data visualization tools that allow operational data analytics. The SQL Analytics solution provides several predefined elastic pool and database monitoring and alerting views and queries. Azure Monitor logs also provides a custom view designer.

OMS workspaces are now referred to as Log Analytics workspaces. Log Analytics workspaces and analytics solutions open in the Azure portal. The Azure portal is the newer access point, but it might be what's behind the Operations Management Suite portal in some areas.

### Create performance diagnostic data by simulating a workload on your tenants

1. In the PowerShell ISE, open ..\WingtipTicketsSaaS-MultiTenantDb-master\Learning Modules\Performance Monitoring and Management\Demo-PerformanceMonitoringAndManagement.ps1. Keep this script open

because you might want to run several of the load generation scenarios during this tutorial.

2. If you haven't done so already, provision a batch of tenants to make the monitoring context more interesting. This process takes a few minutes.
  - a. Set **\$DemoScenario = 1**, *Provision a batch of tenants.*
  - b. To run the script and deploy an additional 17 tenants, press F5.
3. Now start the load generator to run a simulated load on all the tenants.
  - a. Set **\$DemoScenario = 2**, *Generate normal intensity load (approximately 30 DTU).*
  - b. To run the script, press F5.

## Get the Wingtip Tickets SaaS database-per-tenant application scripts

The Wingtip Tickets SaaS multitenant database scripts and application source code are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. For steps to download and unblock the Wingtip Tickets PowerShell scripts, see the [general guidance](#).

## Install and configure Log Analytics workspace and the Azure SQL Analytics solution

Azure Monitor is a separate service that must be configured. Azure Monitor logs collects log data, telemetry, and metrics in a Log Analytics workspace. Just like other resources in Azure, a Log Analytics workspace must be created. The workspace doesn't need to be created in the same resource group as the applications it monitors. Doing so often makes the most sense though. For the Wingtip Tickets app, use a single resource group to make sure the workspace is deleted with the application.

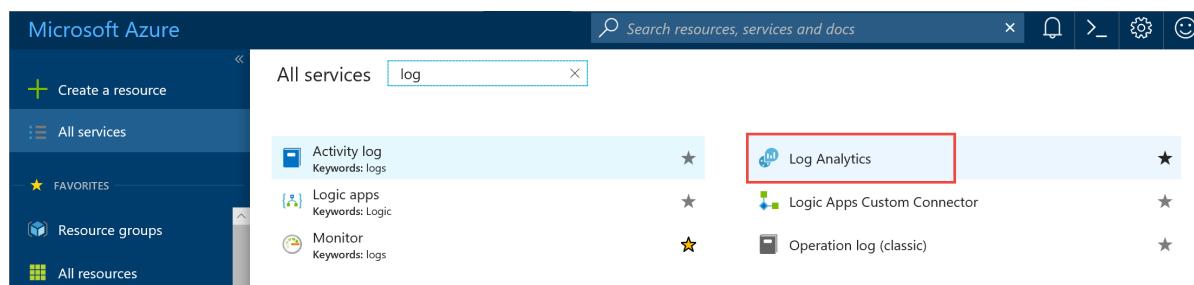
1. In the PowerShell ISE, open ..\WingtipTicketsSaaS-MultiTenantDb-master\Learning Modules\Performance Monitoring and Management\Log Analytics\Demo-LogAnalytics.ps1.
2. To run the script, press F5.

Now you can open Azure Monitor logs in the Azure portal. It takes a few minutes to collect telemetry in the Log Analytics workspace and to make it visible. The longer you leave the system gathering diagnostic data, the more interesting the experience is.

## Use Log Analytics workspace and the SQL Analytics solution to monitor pools and databases

In this exercise, open Log Analytics workspace in the Azure portal to look at the telemetry gathered for the databases and pools.

1. Browse to the [Azure portal](#). Select **All services** to open Log Analytics workspace. Then, search for Log Analytics.



2. Select the workspace named `wtploganalytics-<user>`.

3. Select **Overview** to open the log analytics solution in the Azure portal.

The screenshot shows the Microsoft Azure portal interface for a Log Analytics workspace named "wtploganalytics-sstein5". The left sidebar contains a vertical list of icons and links, including "OMS Workspace", "Activity log", "Access control (IAM)", "Tags", "Diagnose and solve problems", "SETTINGS" (with "Locks" and "Automation script"), "GENERAL" (with "Quick Start", "Overview" which is highlighted with a red box, "Saved searches", "Log Search", "Solutions", and "Pricing tier"), and a search bar at the top. The main content area on the right displays "Essentials" information (Resource group: sstein5, Status: Active, Location: West Central US, Subscription name: SQL DB Content, Subscription ID) and a "Management" section with a "Overview" button (also highlighted with a red box). Below this is a "Pricing tier" section showing "Free" usage for WTPLOGANALYTICS-SSTEIN5... with a daily limit of 500 MB and 7 days of data retention.

**IMPORTANT**

It might take a couple of minutes before the solution is active.

4. Select the **Azure SQL Analytics** tile to open it.

The screenshot shows the Microsoft Azure Overview page for the resource group 'wtplogalytics-sstein5'. On the left, there's a vertical sidebar with various icons for different services like Storage, Compute, and Network. The main area displays a summary for 'Azure SQL Analytics (Preview)'. It shows a large number '23' representing 'Total SQL Azure Databases' and a smaller number '1' representing 'Total SQL Azure Elastic Pools'. A red box highlights this summary section.

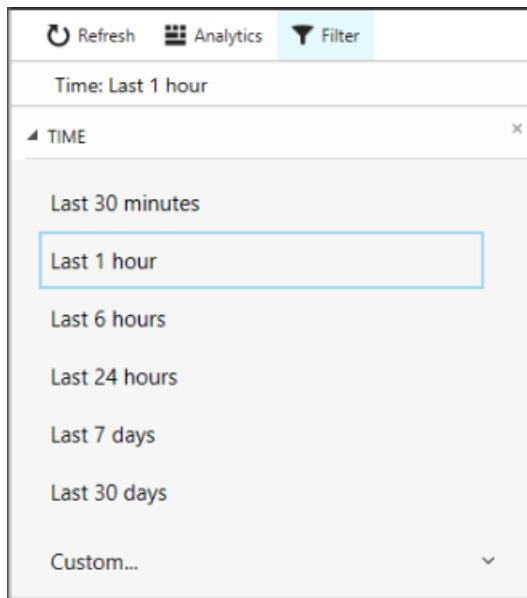
5. The views in the solution scroll sideways, with their own inner scroll bar at the bottom. Refresh the page if necessary.
6. To explore the summary page, select the tiles or individual databases to open a drill-down explorer.

This screenshot provides a detailed look at the 'Azure SQL Analytics (Preview)' summary page. At the top, it shows the navigation path: Home > Log Analytics > wtplogalytics-bg1 > Overview > Azure SQL Analytics (Preview). Below this is a toolbar with Refresh, Analytics, and Filter buttons. The time range is set to 'Last 24 hours'. The main content area is divided into several sections:

- DATABASE FLEET OVERVIEW:** A table showing database metrics. The first few rows are:
 

DATABASE	METRIC	VALUE
tenants1-dpt-bg1.balsamblues...	CPU	8.72
tenants1-dpt-bg1.limetreetrak...	CPU	8.25
tenants1-dpt-bg1.mahoganyso...	CPU	7.39
tenants1-dpt-bg1.blueoakjazzcl...	CPU	7.22
- Number of resources per utilization bucket:** A bar chart showing the distribution of resources across utilization buckets. The x-axis represents time from 6:00 PM to 12:00 PM, and the y-axis represents the count of resources, ranging from 0 to 25.
- Query duration in seconds:** A line chart showing the distribution of query durations over time. The x-axis shows hours from 6:00 PM to 12:00 PM, and the y-axis shows duration in seconds from 0 to 0.4.
- Total time queries spent waiting per type:** A stacked bar chart showing the total time spent waiting for queries by type. The x-axis shows hours from 6:00 PM to 12:00 PM, and the y-axis shows waiting time in seconds from 0 to 1.
- Total time DBs spent waiting per type:** A stacked bar chart showing the total time spent waiting for databases by type. The x-axis shows hours from 6:00 PM to 12:00 PM, and the y-axis shows waiting time in seconds from 0 to 1k.
- Requires configuration:** A note stating 'Please make sure that SQLInsights Diagnostics log is enabled.' with a link to learn more about SQL Diagnostics logs.

7. Change the filter setting to modify the time range. For this tutorial, select **Last 1 hour**.



8. Select an individual database to explore the query usage and metrics for that database.

The screenshot shows the Azure SQL Analytics (Preview) interface for a specific database. The top navigation bar includes 'Home', 'Log Analytics', 'Database', 'Analytics', 'Filter', and 'Refresh'. The main content area is divided into three sections:

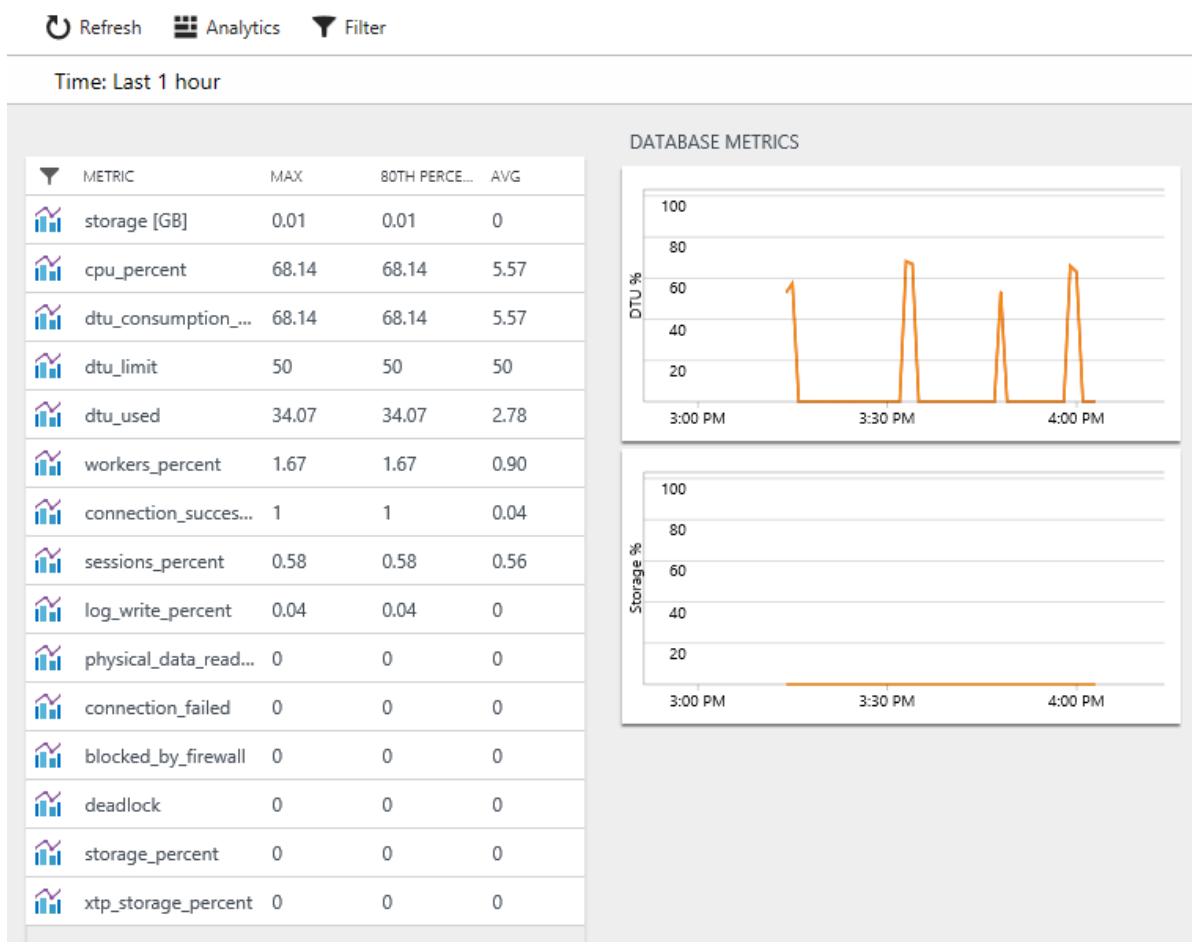
- RESOURCE INFO:** Shows 'Subscription name: 112900e0-ad0e-4ab8-aea2-b09a07253f30' and 'Server name: tenants1-dpt-bg1'.
- INSIGHTS SUMMARY:** Displays a message: 'Requires configuration. Please make sure that SQLInsights Diagnostics log is enabled.' with a link to 'Learn more about SQL Diagnostics logs'. Below this is a button 'View full insights report'.
- QUERIES:** A chart showing 'QueryWait (s)' over time from 3:00 PM to 4:00 PM. The chart has two bars: one at 3:30 PM with a value of approximately 0.01, and one at 3:50 PM with a value of approximately 0.02. Below the chart is a table of query metrics:

QUERY	METRIC	MAX (s)	Avg (%)	Dominant Wait	Max (%)	Avg (%)	EXCS
0xAB1EF762A66266...	Duration	0	0	MEMORY	0	0	480
0xC824056576DF...	Duration	0	0		0	0	6
0x850035D0EAFD...	Duration	0	0		0	0	2
0x896E913B7EA1...	Duration	0	0		0	0	480
0xF562499F92BC78...	Duration	0	0		0	0	480
0x6707411894A1...	Duration	0	0		0	0	240
0x9C5D46A6BE65...	Duration	0	0		0	0	480

9. To see usage metrics, scroll the analytics page to the right.

Home > Log Analytics > wtploganalytics-bg1 > Overview > Azure SQL Analytics (Preview) > Database > Server > Elastic Pool > Database

**Database**  
wtploganalytics-bg1



10. Scroll the analytics page to the left, and select the server tile in the **Resource Info** list.

**RESOURCE INFO**

Subscription name:  
112900e0-ad0e-4ab8-aea2-b09a07253f30

Server name:  
tenants1-dpt-bg1

Database name:  
balsambluesclub

A page opens that shows the pools and databases on the server.

Server  
wtploganalytics-bg1

Refresh Analytics Filter

Time: Last 1 hour

RESOURCE INFO				ELASTIC POOLS			DATABASES			
		POOL	METRIC		POOL	METRIC		POOL	METRIC	
Subscription name: 112900e0-ad0e-4ab8-aea2-b09a07253f30	<input type="checkbox"/>	pool1	CPU	78.86	limetreetrack	CPU	8.63	balsambluesclub	CPU	8.31
Server name: tenants1-dpt-bg1	<input type="checkbox"/>				hornbeamhiphop	CPU	8.23	mahoganysoccer	CPU	8.12
					dogwooddojo	CPU	7.94	osageopera	CPU	7.84
					sycamoresymphony	CPU	7.63	cottonwoodconcerthall	CPU	7.50
					sorrelsoccer	CPU	7.29	mangrovesoccerclub	CPU	6.91
					greenmapleracing	CPU	6.89	papayaplayers	CPU	6.74
					juniperjammersjazz	CPU	6.32	staranisejudo	CPU	5.77
					foxtailrock	CPU	5.70			

< 1 of 1 >

< 1 of 2 >

11. Select a pool. On the pool page that opens, scroll to the right to see the pool metrics.

Home > Log Analytics > wtploganalytics-bg1 > Overview > Azure SQL Analytics (Preview) > Database > Server > Elastic Pool

Elastic Pool  
wtploganalytics-bg1

Refresh Analytics Filter

Time: Last 1 hour

METRIC	MAX	80TH PERCE...	AVG
storage_limit [GB]	50	50	37.50
storage_used [GB]	0.14	0.14	0.11
cpu_percent	100	100	45.48
dtu_consumption_...	100	100	45.48
eDTU_used	50	50	22.74
eDTU_limit	50	50	50
workers_percent	5	5	1.13
log_write_percent	1.78	1.78	0.01
sessions_percent	0.79	0.79	0.78
storage_percent	0.28	0.28	0.28
physical_data_read...	0	0	0
xtp_storage_percent	0	0	0

ELASTIC POOL METRICS

Storage %

12. Back in the Log Analytics workspace, select **OMS Portal** to open the workspace there.

The screenshot shows the Azure Log Analytics workspace settings page. The left sidebar includes links for OMS Workspace, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. Under SETTINGS, there are links for Locks, Automation script, and Advanced settings. The main content area has tabs for Essentials, Management, and Metrics. The Management tab is active, showing four buttons: Overview, Log Search, OMS Portal (which is highlighted with a red box), and View Designer.

In the Log Analytics workspace, you can explore the log and metric data further.

Monitoring and alerting in Azure Monitor logs are based on queries over the data in the workspace, unlike the alerting defined on each resource in the Azure portal. By basing alerts on queries, you can define a single alert that looks over all databases, rather than defining one per database. Queries are limited only by the data available in the workspace.

For more information on how to use Azure Monitor logs to query and set alerts, see [Work with alert rules in Azure Monitor logs](#).

Azure Monitor logs for SQL Database charges based on the data volume in the workspace. In this tutorial, you created a free workspace, which is limited to 500 MB per day. After that limit is reached, data is no longer added to the workspace.

## Next steps

In this tutorial you learned how to:

- Install and configure Azure Monitor logs.
- Use Azure Monitor logs to monitor pools and databases.

Try the [Tenant analytics tutorial](#).

## Additional resources

- [Additional tutorials that build on the initial Wingtip Tickets SaaS database-per-tenant application deployment](#)
- [Azure Monitor logs](#)

# Restore a single tenant with a database-per-tenant SaaS application

11/7/2019 • 6 minutes to read • [Edit Online](#)

The database-per-tenant model makes it easy to restore a single tenant to a prior point in time without affecting other tenants.

In this tutorial, you learn two data recovery patterns:

- Restore a database into a parallel database (side by side).
- Restore a database in place, replacing the existing database.

Restore into a parallel database	This pattern can be used for tasks such as review, auditing, and compliance to allow a tenant to inspect their data from an earlier point. The tenant's current database remains online and unchanged.
Restore in place	This pattern is typically used to recover a tenant to an earlier point, after a tenant accidentally deletes or corrupts data. The original database is taken off line and replaced with the restored database.

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip SaaS app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip SaaS application](#).
- Azure PowerShell is installed. For details, see [Get started with Azure PowerShell](#).

## Introduction to the SaaS tenant restore patterns

There are two simple patterns for restoring an individual tenant's data. Because tenant databases are isolated from each other, restoring one tenant has no impact on any other tenant's data. The Azure SQL Database point-in-time-restore (PITR) feature is used in both patterns. PITR always creates a new database.

- **Restore in parallel:** In the first pattern, a new parallel database is created alongside the tenant's current database. The tenant is then given read-only access to the restored database. The restored data can be reviewed and potentially used to overwrite current data values. It's up to the app designer to determine how the tenant accesses the restored database and what options for recovery are provided. Simply allowing the tenant to review their data at an earlier point might be all that's required in some scenarios.
- **Restore in place:** The second pattern is useful if data was lost or corrupted and the tenant wants to revert to an earlier point. The tenant is taken off line while the database is restored. The original database is deleted, and the restored database is renamed. The backup chain of the original database remains accessible after the deletion, so you can restore the database to an earlier point in time, if necessary.

If the database uses [active geo-replication](#) and restoring in parallel, we recommend that you copy any required data from the restored copy into the original database. If you replace the original database with the restored database, you need to reconfigure and resynchronize geo-replication.

# Get the Wingtip Tickets SaaS database-per-tenant application scripts

The Wingtip Tickets SaaS Multitenant Database scripts and application source code are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. For steps to download and unlock the Wingtip Tickets SaaS scripts, see the [general guidance](#).

## Before you start

When a database is created, it can take 10 to 15 minutes before the first full backup is available to restore from. If you just installed the application, you might need to wait for a few minutes before you try this scenario.

## Simulate a tenant accidentally deleting data

To demonstrate these recovery scenarios, first "accidentally" delete an event in one of the tenant databases.

### Open the Events app to review the current events

1. Open the Events Hub (<http://events.wtp.<user>.trafficmanager.net>), and select **Contoso Concert Hall**.

The screenshot shows the 'Events Hub' section of the Wingtip Tickets Platform. At the top, there's a search bar labeled 'search'. Below it is a table with a header 'Venues' and a red arrow pointing down to its right. The table lists ten venue names: Balsam Blues Club, Blue Oak Jazz Club, Bushwillow Blues, Contoso Concert Hall (which is highlighted with a red border), Cottonwood Concert Hall, Dogwood Dojo, Fabrikam Jazz Club, Foxtail Rock, Hackberry Hitters, and Hornbeam HipHop. At the bottom of the table, there are page navigation links: '1 | 2 | 3 | End'.

Venues
Balsam Blues Club
Blue Oak Jazz Club
Bushwillow Blues
Contoso Concert Hall
Cottonwood Concert Hall
Dogwood Dojo
Fabrikam Jazz Club
Foxtail Rock
Hackberry Hitters
Hornbeam HipHop

2. Scroll the list of events, and make a note of the last event in the list.

Next | A Musical Journey >

		Tickets
MAY 26 FRI	Handel's Messiah Contoso Symphony	Tickets
MAY 29 MON	Moonlight Serenade Contoso Quartet	Tickets
JUN 1 THU	Seriously Strauss Julie von Strauss Septet	Tickets

**mytickets**  
Update your list of favorites and never miss an event!  
[Sign In](#)

Contoso Concert Hall

POWERED BY THE WINGTIP TICKETS PLATFORM

LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE  
Running on Azure SQL Database. © 2017 Microsoft  
server: tenants1-sjs2.database.windows.net  
database: contosoconcerthall\_old tenant id: 5D172CF1

### "Accidentally" delete the last event

- In the PowerShell ISE, open ...\\Learning Modules\\Business Continuity and Disaster Recovery\\RestoreTenant\\Demo-RestoreTenant.ps1, and set the following value:
  - \$DemoScenario = 1, Delete last event (with no ticket sales).**
- Press F5 to run the script and delete the last event. The following confirmation message appears:

```
Deleting last unsold event from Contoso Concert Hall ...
Deleted event 'Seriously Strauss' from Contoso Concert Hall venue.
```

- The Contoso events page opens. Scroll down and verify that the event is gone. If the event is still in the list, select **Refresh** and verify that it's gone.

Next | A Night at the Opera >

MAY 23 TUE	The 1812 Overture Contoso Symphony	Tickets
MAY 26 FRI	Handel's Messiah Contoso Symphony	Tickets
MAY 29 MON	Moonlight Serenade Contoso Quartet	Tickets
<b>Contoso Concert Hall</b> <small>POWERED BY THE WINGTIP TICKETS PLATFORM</small>		

**my|tickets**  
 Update your list of favorites and never miss an event!

[Sign In](#)

LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE  
 Running on Azure SQL Database. © 2017 Microsoft  
 server: tenants1-sjs2.database.windows.net  
 database: contosoconcerthall tenant id: F5C9F146

## Restore a tenant database in parallel with the production database

This exercise restores the Contoso Concert Hall database to a point in time before the event was deleted. This scenario assumes that you want to review the deleted data in a parallel database.

The *Restore-TenantInParallel.ps1* script creates a parallel tenant database named *ContosoConcertHall\_old*, with a parallel catalog entry. This pattern of restore is best suited for recovering from a minor data loss. You also can use this pattern if you need to review data for compliance or auditing purposes. It's the recommended approach when you use [active geo-replication](#).

1. Complete the [Simulate a tenant accidentally deleting data](#) section.
2. In the PowerShell ISE, open ...\\Learning Modules\\Business Continuity and Disaster Recovery\\RestoreTenant\\*Demo-RestoreTenant.ps1*.
3. Set **\$DemoScenario = 2**, *Restore tenant in parallel*.
4. To run the script, press F5.

The script restores the tenant database to a point in time before you deleted the event. The database is restored to a new database named *ContosoConcertHall\_old*. The catalog metadata that exists in this restored database is deleted, and then the database is added to the catalog by using a key constructed from the

*ContosoConcertHall\_old* name.

The demo script opens the events page for this new tenant database in your browser. Note from the URL [http://events.wingtip-dpt.&lt;user&gt;.trafficmanager.net/contosoconcerthall\\_old](http://events.wingtip-dpt.&lt;user&gt;.trafficmanager.net/contosoconcerthall_old) that this page shows data from the restored database where *\_old* is added to the name.

Scroll the events listed in the browser to confirm that the event deleted in the previous section was restored.

Exposing the restored tenant as an additional tenant, with its own Events app, is unlikely to be how you provide a tenant access to restored data. It serves to illustrate the restore pattern. Typically, you give read-only access to the old data and retain the restored database for a defined period. In the sample, you can delete the restored tenant entry after you're finished by running the *Remove restored tenant* scenario.

1. Set **\$DemoScenario = 4, Remove restored tenant.**
2. To run the script, press F5.
3. The *ContosoConcertHall\_old* entry is now deleted from the catalog. Close the events page for this tenant in your browser.

## Restore a tenant in place, replacing the existing tenant database

This exercise restores the Contoso Concert Hall tenant to a point before the event was deleted. The *Restore-TenantInPlace* script restores a tenant database to a new database and deletes the original. This restore pattern is best suited to recovering from serious data corruption, and the tenant might have to accommodate significant data loss.

1. In the PowerShell ISE, open the **Demo-RestoreTenant.ps1** file.
2. Set **\$DemoScenario = 5, Restore tenant in place.**
3. To run the script, press F5.

The script restores the tenant database to a point before the event was deleted. It first takes the Contoso Concert Hall tenant off line to prevent further updates. Then, a parallel database is created by restoring from the restore point. The restored database is named with a time stamp to make sure the database name doesn't conflict with the existing tenant database name. Next, the old tenant database is deleted, and the restored database is renamed to the original database name. Finally, Contoso Concert Hall is brought online to allow the app access to the restored database.

You successfully restored the database to a point in time before the event was deleted. When the **Events** page opens, confirm that the last event was restored.

After you restore the database, it takes another 10 to 15 minutes before the first full backup is available to restore from again.

## Next steps

In this tutorial, you learned how to:

- Restore a database into a parallel database (side by side).
- Restore a database in place.

Try the [Manage tenant database schema](#) tutorial.

## Additional resources

- [Additional tutorials that build on the Wingtip SaaS application](#)
- [Overview of business continuity with Azure SQL Database](#)
- [Learn about SQL Database backups](#)



# Manage schema in a SaaS application using the database-per-tenant pattern with Azure SQL Database

11/7/2019 • 6 minutes to read • [Edit Online](#)

As a database application evolves, changes inevitably need to be made to the database schema or reference data. Database maintenance tasks are also needed periodically. Managing an application that uses the database per tenant pattern requires that you apply these changes or maintenance tasks across a fleet of tenant databases.

This tutorial explores two scenarios - deploying reference data updates for all tenants, and rebuilding an index on the table containing the reference data. The [Elastic jobs](#) feature is used to execute these actions on all tenant databases, and on the template database used to create new tenant databases.

In this tutorial you learn how to:

- Create a job agent
- Cause T-SQL jobs to be run on all tenant databases
- Update reference data in all tenant databases
- Create an index on a table in all tenant databases

To complete this tutorial, make sure the following prerequisites are met:

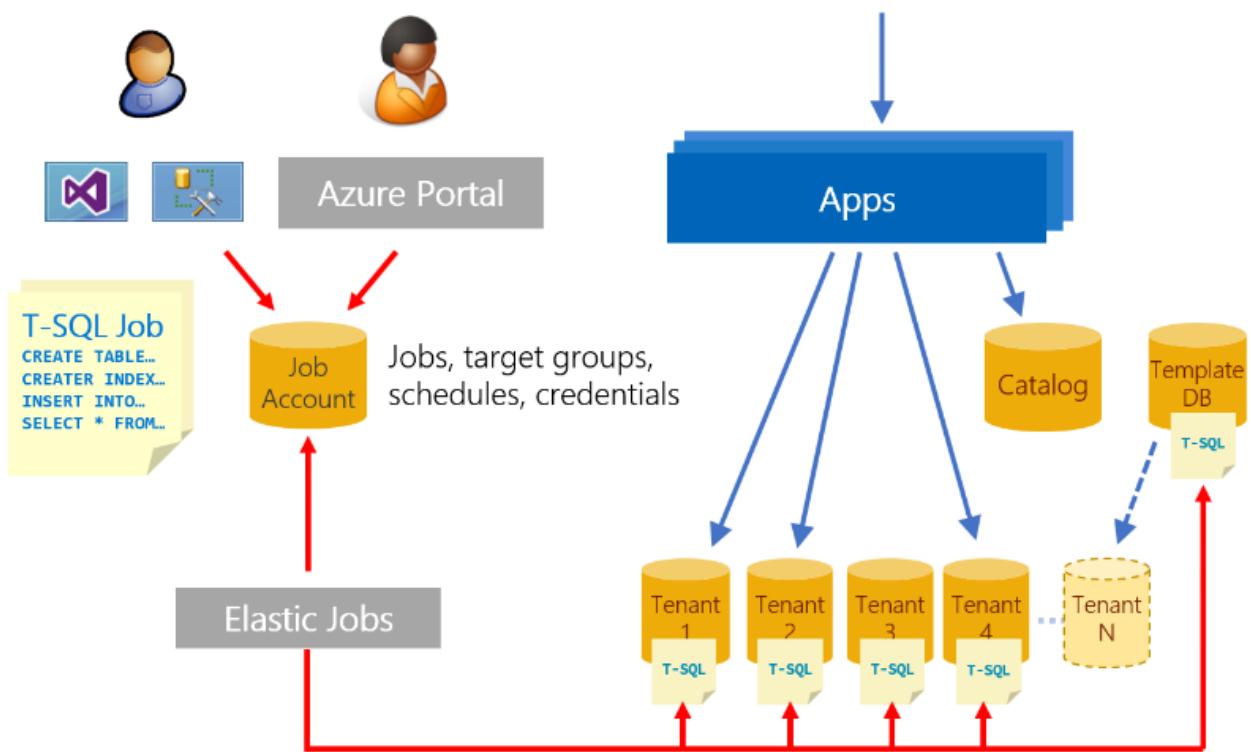
- The Wingtip Tickets SaaS Database Per Tenant app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS database per tenant application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- The latest version of SQL Server Management Studio (SSMS) is installed. [Download and Install SSMS](#)

## NOTE

This tutorial uses features of the SQL Database service that are in a limited preview (Elastic Database jobs). If you wish to do this tutorial, provide your subscription ID to SaaSFeedback@microsoft.com with subject=Elastic Jobs Preview. After you receive confirmation that your subscription has been enabled, [download and install the latest pre-release jobs cmdlets](#). This preview is limited, so contact SaaSFeedback@microsoft.com for related questions or support.

## Introduction to SaaS schema management patterns

The database per tenant pattern isolates tenant data effectively, but increases the number of databases to manage and maintain. [Elastic Jobs](#) facilitates administration and management of SQL databases. Jobs enable you to securely and reliably, run tasks (T-SQL scripts) against a group of databases. Jobs can deploy schema and common reference data changes across all tenant databases in an application. Elastic Jobs can also be used to maintain a *template* database used to create new tenants, ensuring it always has the latest schema and reference data.



## Elastic Jobs limited preview

There's a new version of Elastic Jobs that is now an integrated feature of Azure SQL Database. This new version of Elastic Jobs is currently in limited preview. This limited preview currently supports using PowerShell to create a job agent, and T-SQL to create and manage jobs.

### NOTE

This tutorial uses features of the SQL Database service that are in a limited preview (Elastic Database jobs). If you wish to do this tutorial, provide your subscription ID to SaaSFeedback@microsoft.com with subject=Elastic Jobs Preview. After you receive confirmation that your subscription has been enabled, [download and install the latest pre-release jobs cmdlets](#). This preview is limited, so contact SaaSFeedback@microsoft.com for related questions or support.

## Get the Wingtip Tickets SaaS database per tenant application scripts

The application source code and management scripts are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

## Create a job agent database and new job agent

This tutorial requires you use PowerShell to create a job agent and its backing job agent database. The job agent database holds job definitions, job status, and history. Once the job agent and its database are created, you can create and monitor jobs immediately.

1. In **PowerShell ISE**, open ...\\Learning Modules\\Schema Management\\*Demo-SchemaManagement.ps1*.
2. Press **F5** to run the script.

The *Demo-SchemaManagement.ps1* script calls the *Deploy-SchemaManagement.ps1* script to create a SQL database named *osagent* on the catalog server. It then creates the job agent, using the database as a parameter.

## Create a job to deploy new reference data to all tenants

In the Wingtip Tickets app, each tenant database includes a set of supported venue types. Each venue is of a specific venue type, which defines the kind of events that can be hosted, and determines the background image used in the app. For the application to support new kinds of events, this reference data must be updated and new venue types added. In this exercise, you deploy an update to all the tenant databases to add two additional venue types: *Motorcycle Racing* and *Swimming Club*.

First, review the venue types included in each tenant database. Connect to one of the tenant databases in SQL Server Management Studio (SSMS) and inspect the *VenueTypes* table. You can also query this table in the Query editor in the Azure portal, accessed from the database page.

1. Open SSMS and connect to the tenant server: *tenants1-dpt-<user>.database.windows.net*
2. To confirm that *Motorcycle Racing* and *Swimming Club* **are not** currently included, browse to the *contosoconcerthall* database on the *tenants1-dpt-<user>* server and query the *VenueTypes* table.

Now let's create a job to update the *VenueTypes* table in all the tenant databases to add the new venue types.

To create a new job, you use a set of jobs system stored procedures created in the *jobagent* database when the job agent was created.

1. In SSMS, connect to the catalog server: *catalog-dpt-<user>.database.windows.net* server
2. In SSMS, open the file ...\\Learning Modules\\Schema Management\\DeployReferenceData.sql
3. Modify the statement: SET @wtpUser = <user> and substitute the User value used when you deployed the Wingtip Tickets SaaS Database Per Tenant app
4. Ensure you are connected to the *jobagent* database and press **F5** to run the script

Observe the following elements in the *DeployReferenceData.sql* script:

- **sp\_add\_target\_group** creates the target group name *DemoServerGroup*.
- **sp\_add\_target\_group\_member** is used to define the set of target databases. First the *tenants1-dpt-<user>* server is added. Adding the server as a target causes the databases in that server at the time of job execution to be included in the job. Then the *basetenantdb* database and the *adhocreporting* database (used in a later tutorial) are added as targets.
- **sp\_add\_job** creates a job named *Reference Data Deployment*.
- **sp\_add\_jobstep** creates the job step containing T-SQL command text to update the reference table, *VenueTypes*.
- The remaining views in the script display the existence of the objects and monitor job execution. Use these queries to review the status value in the **lifecycle** column to determine when the job has finished on all the target databases.

Once the script has completed, you can verify the reference data has been updated. In SSMS, browse to the *contosoconcerthall* database on the *tenants1-dpt-<user>* server and query the *VenueTypes* table. Check that *Motorcycle Racing* and *Swimming Club* **are** now present.

## Create a job to manage the reference table index

This exercise uses a job to rebuild the index on the reference table primary key. This is a typical database maintenance operation that might be done after loading large amounts of data.

Create a job using the same jobs 'system' stored procedures.

1. Open SSMS and connect to the *catalog-dpt-<user>.database.windows.net* server
2. Open the file ...\\Learning Modules\\Schema Management\\OnlineReindex.sql
3. Right click, select Connection, and connect to the *catalog-dpt-<user>.database.windows.net* server, if not already connected
4. Ensure you are connected to the *jobagent* database and press **F5** to run the script

Observe the following elements in the *OnlineReindex.sql* script:

- **sp\_add\_job** creates a new job called “Online Reindex PK\_\_VenueTyp\_\_265E44FD7FD4C885”
- **sp\_add\_jobstep** creates the job step containing T-SQL command text to update the index
- The remaining views in the script monitor job execution. Use these queries to review the status value in the **lifecycle** column to determine when the job has successfully finished on all target group members.

## Next steps

In this tutorial you learned how to:

- Create a job agent to run across T-SQL jobs multiple databases
- Update reference data in all tenant databases
- Create an index on a table in all tenant databases

Next, try the [Ad hoc reporting tutorial](#) to explore running distributed queries across tenant databases.

## Additional resources

- [Additional tutorials that build upon the Wingtip Tickets SaaS Database Per Tenant application deployment](#)
- [Managing scaled-out cloud databases](#)

# Cross-tenant reporting using distributed queries

11/7/2019 • 8 minutes to read • [Edit Online](#)

In this tutorial, you run distributed queries across the entire set of tenant databases for reporting. These queries can extract insights buried in the day-to-day operational data of the Wingtip Tickets SaaS tenants. To do this, you deploy an additional reporting database to the catalog server and use Elastic Query to enable distributed queries.

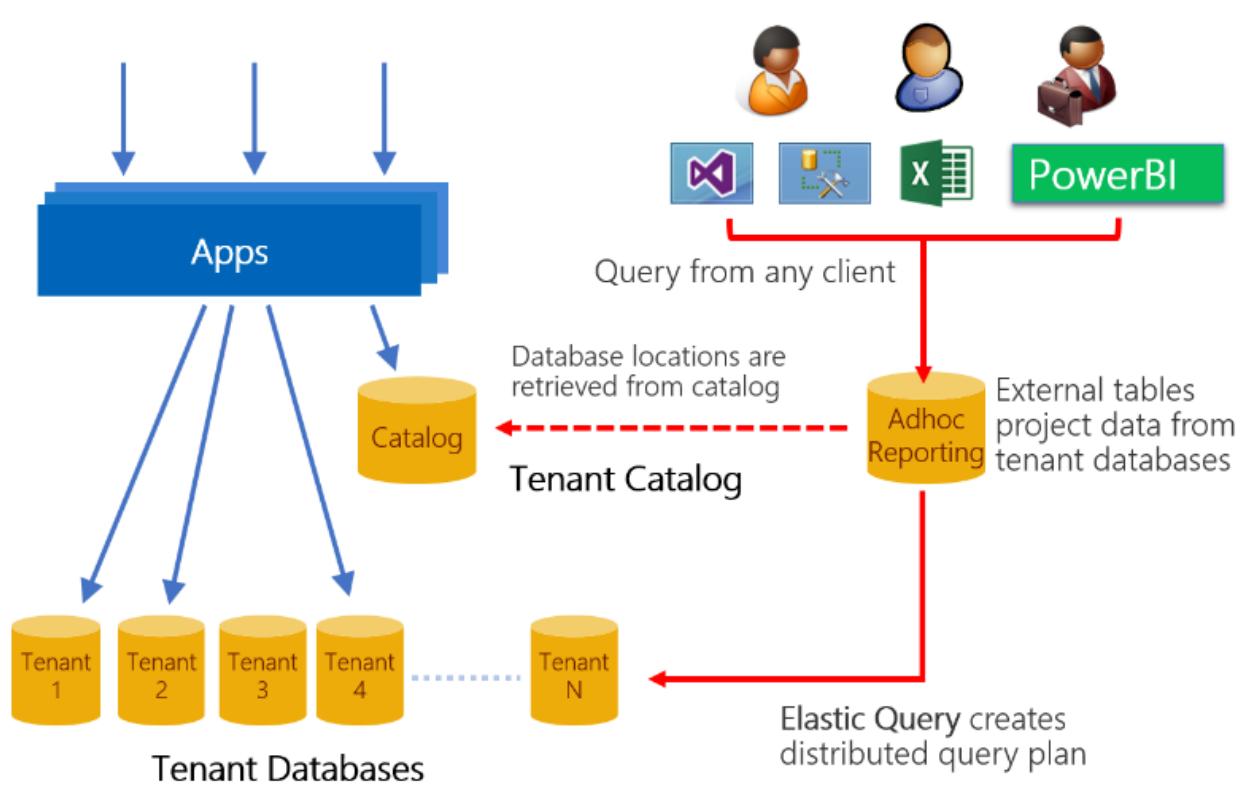
In this tutorial you learn:

- How to deploy an reporting database
- How to run distributed queries across all tenant databases
- How global views in each database can enable efficient querying across tenants

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS Database Per Tenant app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Database Per Tenant application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- SQL Server Management Studio (SSMS) is installed. To download and install SSMS, see [Download SQL Server Management Studio \(SSMS\)](#).

## Cross-tenant reporting pattern



One opportunity with SaaS applications is to use the vast amount of tenant data stored in the cloud to gain insights into the operation and usage of your application. These insights can guide feature development, usability improvements, and other investments in your apps and services.

Accessing this data in a single multi-tenant database is easy, but not so easy when distributed at scale across potentially thousands of databases. One approach is to use [Elastic Query](#), which enables querying across a

distributed set of databases with common schema. These databases can be distributed across different resource groups and subscriptions, but need to share a common login. Elastic Query uses a single *head* database in which external tables are defined that mirror tables or views in the distributed (tenant) databases. Queries submitted to this head database are compiled to produce a distributed query plan, with portions of the query pushed down to the tenant databases as needed. Elastic Query uses the shard map in the catalog database to determine the location of all tenant databases. Setup and query of the head database are straightforward using standard [Transact-SQL](#), and support querying from tools like Power BI and Excel.

By distributing queries across the tenant databases, Elastic Query provides immediate insight into live production data. As Elastic Query pulls data from potentially many databases, query latency can be higher than equivalent queries submitted to a single multi-tenant database. Design queries to minimize the data that is returned to the head database. Elastic Query is often best suited for querying small amounts of real-time data, as opposed to building frequently used or complex analytics queries or reports. If queries don't perform well, look at the [execution plan](#) to see what part of the query is pushed down to the remote database and how much data is being returned. Queries that require complex aggregation or analytical processing may be better handles by extracting tenant data into a database or data warehouse optimized for analytics queries. This pattern is explained in the [tenant analytics tutorial](#).

## Get the Wingtip Tickets SaaS Database Per Tenant application scripts

The Wingtip Tickets SaaS Multi-tenant Database scripts and application source code are available in the [WingtipTicketsSaaS-DbPerTenant](#) GitHub repo. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

### Create ticket sales data

To run queries against a more interesting data set, create ticket sales data by running the ticket-generator.

1. In the *PowerShell ISE*, open the ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\*Demo-AdhocReporting.ps1* script and set the following value:
  - **\$DemoScenario = 1, Purchase tickets for events at all venues.**
2. Press **F5** to run the script and generate ticket sales. While the script is running, continue the steps in this tutorial. The ticket data is queried in the *Run ad hoc distributed queries* section, so wait for the ticket generator to complete.

### Explore the global views

In the Wingtip Tickets SaaS Database Per Tenant application, each tenant is given a database. Thus, the data contained in the database tables is scoped to the perspective of a single tenant. However, when querying across all databases, it's important that Elastic Query can treat the data as if it is part of a single logical database sharded by tenant.

To simulate this pattern, a set of 'global' views are added to the tenant database that project a tenant ID into each of the tables that are queried globally. For example, the *VenueEvents* view adds a computed *VenuelId* to the columns projected from the *Events* table. Similarly, the *VenueTicketPurchases* and *VenueTickets* views add a computed *VenuelId* column projected from their respective tables. These views are used by Elastic Query to parallelize queries and push them down to the appropriate remote tenant database when a *VenuelId* column is present. This dramatically reduces the amount of data that is returned and results in a substantial increase in performance for many queries. These global views have been pre-created in all tenant databases.

1. Open SSMS and [connect to the tenants1-<USER> server](#).
2. Expand **Databases**, right-click *contosoconcerthall*, and select **New Query**.
3. Run the following queries to explore the difference between the single-tenant tables and the global views:

```
-- The base Venue table, that has no VenueId associated.
SELECT * FROM Venue

-- Notice the plural name 'Venues'. This view projects a VenueId column.
SELECT * FROM Venues

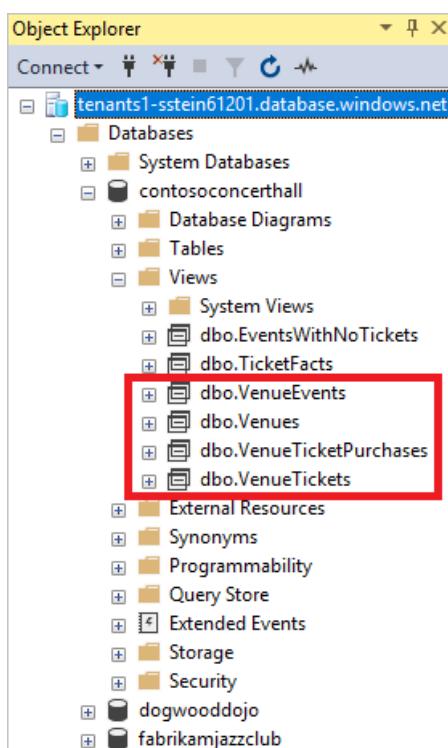
-- The base Events table, which has no VenueId column.
SELECT * FROM Events

-- This view projects the VenueId retrieved from the Venues table.
SELECT * FROM VenueEvents
```

In these views, the *VenueId* is computed as a hash of the Venue name, but any approach could be used to introduce a unique value. This approach is similar to the way the tenant key is computed for use in the catalog.

To examine the definition of the *Venues* view:

1. In **Object Explorer**, expand **contosoconcerthall > Views**:



2. Right-click **dbo.Venues**.
3. Select **Script View as > CREATE To > New Query Editor Window**

Script any of the other *Venue* views to see how they add the *VenueId*.

## Deploy the database used for distributed queries

This exercise deploys the *adhocreporting* database. This is the head database that contains the schema used for querying across all tenant databases. The database is deployed to the existing catalog server, which is the server used for all management-related databases in the sample app.

1. in *PowerShell ISE*, open ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\Demo-*AdhocReporting.ps1*.
2. Set **\$DemoScenario = 2**, Deploy Ad hoc reporting database.
3. Press **F5** to run the script and create the *adhocreporting* database.

In the next section, you add schema to the database so it can be used to run distributed queries.

## Configure the 'head' database for running distributed queries

This exercise adds schema (the external data source and external table definitions) to the *adhocreporting* database to enable querying across all tenant databases.

1. Open SQL Server Management Studio, and connect to the Adhoc Reporting database you created in the previous step. The name of the database is *adhocreporting*.
2. Open ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\ *Initialize-AdhocReportingDB.sql* in SSMS.
3. Review the SQL script and note:

Elastic Query uses a database-scoped credential to access each of the tenant databases. This credential needs to be available in all the databases and should normally be granted the minimum rights required to enable these queries.

```
-- Create login credential, used to access the catalog and remote databases
IF NOT EXISTS (SELECT * FROM sys.database_scoped_credentials WHERE name = 'AdhocQueryDBCred')
 CREATE DATABASE SCOPED CREDENTIAL [AdhocQueryDBCred] WITH IDENTITY = N'developer', SECRET = N'P@ssword1';
GO
```

With the catalog database as the external data source, queries are distributed to all databases registered in the catalog at the time the query runs. As server names are different for each deployment, this script gets the location of the catalog database from the current server (@@servername) where the script is executed.

```
-- Add catalog database as external data source using credential created above
IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'WtpTenantDBs')
BEGIN
 DECLARE @catalogServerName nvarchar(128) = (SELECT @@servername + '.database.windows.net');
 DECLARE @createExternalSource nvarchar(500) =
 N'CREATE EXTERNAL DATA SOURCE [WtpTenantDBs]
 WITH
 (
 TYPE = SHARD_MAP_MANAGER,
 LOCATION = ''' + @catalogServerName + ''',
 DATABASE_NAME = ''tenantcatalog'',
 SHARD_MAP_NAME = ''tenantcatalog'',
 CREDENTIAL = [AdhocQueryDBCred]
);'
 EXEC(@createExternalSource)
END
```

The external tables that reference the global views described in the previous section, and defined with **DISTRIBUTION = SHARDED(VenueId)**. Because each *VenueId* maps to an individual database, this improves performance for many scenarios as shown in the next section.

```
CREATE EXTERNAL TABLE [dbo].[VenueEvents]
(
 [VenueId] INT NOT NULL,
 [EventId] INT NOT NULL,
 [EventName] NVARCHAR (50) NOT NULL,
 [Subtitle] NVARCHAR (50) NULL,
 [Date] DATETIME NOT NULL
)
WITH
(
 DATA_SOURCE = [WtpTenantDBs],
 DISTRIBUTION = SHARDED(VenueId)
);
```

The local table *VenueTypes* that is created and populated. This reference data table is common in all tenant databases, so it can be represented here as a local table and populated with the common data. For some queries, having this table defined in the head database can reduce the amount of data that needs to be moved to the head database.

```

CREATE TABLE [dbo].[VenueTypes]
(
 [VenueType] CHAR(30) NOT NULL,
 [VenueTypeName] NCHAR(30) NOT NULL,
 [EventTypeName] NVARCHAR(30) NOT NULL,
 [EventTypeShortName] NVARCHAR(20) NOT NULL,
 [EventTypeShortNamePlural] NVARCHAR(20) NOT NULL,
 [Language] CHAR(8) NOT NULL,
 PRIMARY KEY CLUSTERED ([VenueType] ASC)
)

```

If you include reference tables in this manner, be sure to update the table schema and data whenever you update the tenant databases.

4. Press **F5** to run the script and initialize the *adhocreporting* database.

Now you can run distributed queries, and gather insights across all tenants!

## Run distributed queries

Now that the *adhocreporting* database is set up, go ahead and run some distributed queries. Include the execution plan for a better understanding of where the query processing is happening.

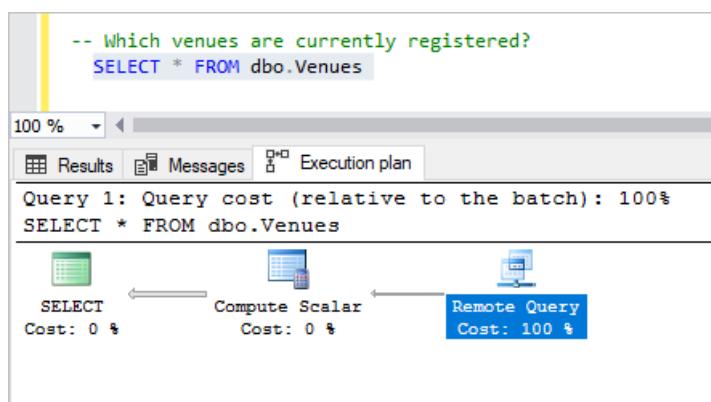
When inspecting the execution plan, hover over the plan icons for details.

Important to note, is that setting **DISTRIBUTION = SHARDED(Venuelid)** when the external data source is defined improves performance for many scenarios. As each *Venuelid* maps to an individual database, filtering is easily done remotely, returning only the data needed.

1. Open ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\Demo-AdhocReportingQueries.sql in SSMS.
2. Ensure you are connected to the **adhocreporting** database.
3. Select the **Query** menu and click **Include Actual Execution Plan**
4. Highlight the *Which venues are currently registered?* query, and press **F5**.

The query returns the entire venue list, illustrating how quick, and easy it is to query across all tenants and return data from each tenant.

Inspect the plan and see that the entire cost is in the remote query. Each tenant database executes the query remotely and returns its venue information to the head database.



5. Select the next query, and press **F5**.

This query joins data from the tenant databases and the local *VenueTypes* table (local, as it's a table in the *adhocreporting* database).

Inspect the plan and see that the majority of cost is the remote query. Each tenant database returns its venue info and performs a local join with the local *VenueTypes* table to display the friendly name.

```
-- Which venues are currently registered?
SELECT * FROM dbo.Venues
GO

-- And what is their venue type?
SELECT VenueName,
 VenueTypeName,
 EventTypeName
 FROM dbo.Venues
 INNER JOIN dbo.VenueTypes ON Venues.VenueType = VenueTypes.VenueType
```

0 % ▾

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT VenueName, VenueTypeName, EventTypeName FROM dbo.Venues INNER JOIN c...

```

graph TD
 A[SELECT Cost: 0 %] --> B[Hash Match (Inner Join) Cost: 7 %]
 B --> C[Clustered Index Scan (Clustered) [VenueTypes].[PK_VenueTyp_265E44F... Cost: 1 %]
 C --> D[Compute Scalar Cost: 0 %]
 D --> E[Remote Query Cost: 92 %]

```

- Now select the *On which day were the most tickets sold?* query, and press **F5**.

This query does a bit more complex joining and aggregation. Most of the processing occurs remotely. Only single rows, containing each venue's daily ticket sale count per day, are returned to the head database.

```
-- On which day were the most tickets sold?
SELECT CAST(PurchaseDate AS DATE) AS TicketPurchaseDate,
 Count(TicketId) AS TicketCount
 FROM VenueTicketPurchases
 INNER JOIN VenueTickets ON (VenueTickets.TicketPurchaseId = VenueTicketPurchases.TicketPurchaseId AND VenueTickets.VenueId = VenueTicketPurchases.VenueId)
 GROUP BY (CAST(PurchaseDate AS DATE))
 ORDER BY TicketCount DESC, TicketPurchaseDate ASC
```

100 % ▾

Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT CAST(PurchaseDate AS DATE) AS TicketPurchaseDate, Count(TicketId) AS TicketCount FROM VenueTicketPurchases...

```

graph LR
 A[SELECT Cost: 0 %] --> B[Sort Cost: 27 %]
 B --> C[Compute Scalar Cost: 0 %]
 C --> D[Stream Aggregate (Aggregate) Cost: 0 %]
 D --> E[Sort Cost: 27 %]
 E --> F[Compute Scalar Cost: 0 %]
 F --> G[Remote Query Cost: 47 %]

```

## Next steps

In this tutorial you learned how to:

- Run distributed queries across all tenant databases
- Deploy a reporting database and define the schema required to run distributed queries.

Now try the [Tenant Analytics tutorial](#) to explore extracting data to a separate analytics database for more complex analytics processing.

## Additional resources

- Additional [tutorials that build upon the Wingtip Tickets SaaS Database Per Tenant application](#)
- [Elastic Query](#)

# Cross-tenant analytics using extracted data - single-tenant app

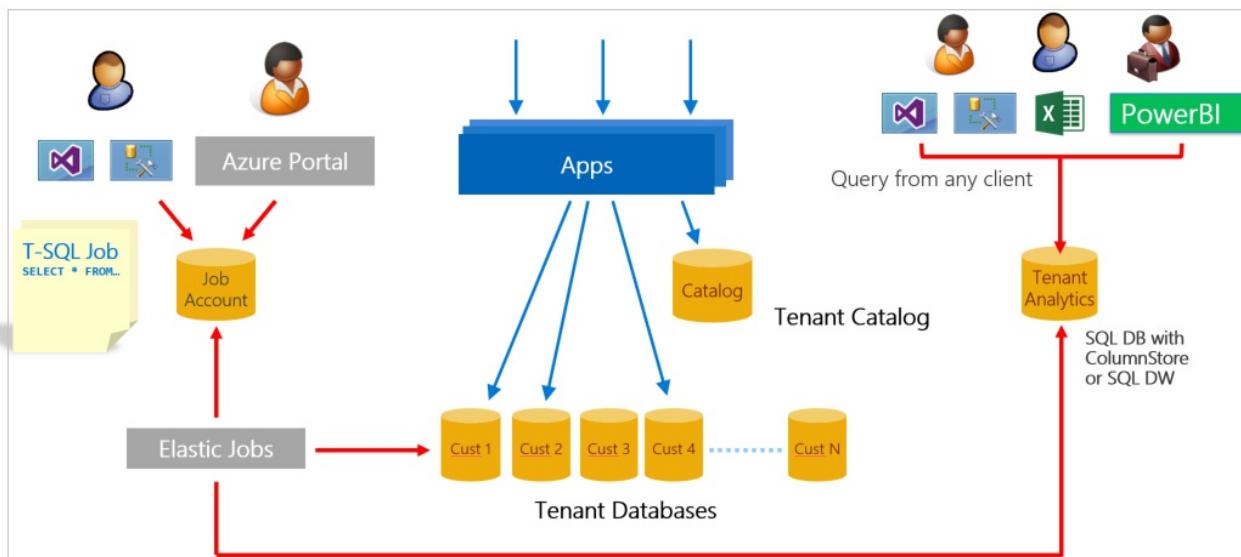
11/7/2019 • 13 minutes to read • [Edit Online](#)

In this tutorial, you walk through a complete analytics scenario for a single tenant implementation. The scenario demonstrates how analytics can enable businesses to make smart decisions. Using data extracted from each tenant database, you use analytics to gain insights into tenant behavior, including their use of the sample Wingtip Tickets SaaS application. This scenario involves three steps:

1. **Extract** data from each tenant database and **Load** into an analytics store.
2. **Transform the extracted data** for analytics processing.
3. Use **business intelligence** tools to draw out useful insights, which can guide decision making.

In this tutorial you learn how to:

- Create the tenant analytics store to extract the data into.
- Use elastic jobs to extract data from each tenant database into the analytics store.
- Optimize the extracted data (reorganize into a star-schema).
- Query the analytics database.
- Use Power BI for data visualization to highlight trends in tenant data and make recommendation for improvements.



## Offline tenant analytics pattern

Multi-tenant SaaS applications typically have a vast amount of tenant data stored in the cloud. This data provides a rich source of insights about the operation and usage of your application, and the behavior of your tenants. These insights can guide feature development, usability improvements, and other investments in the app and platform.

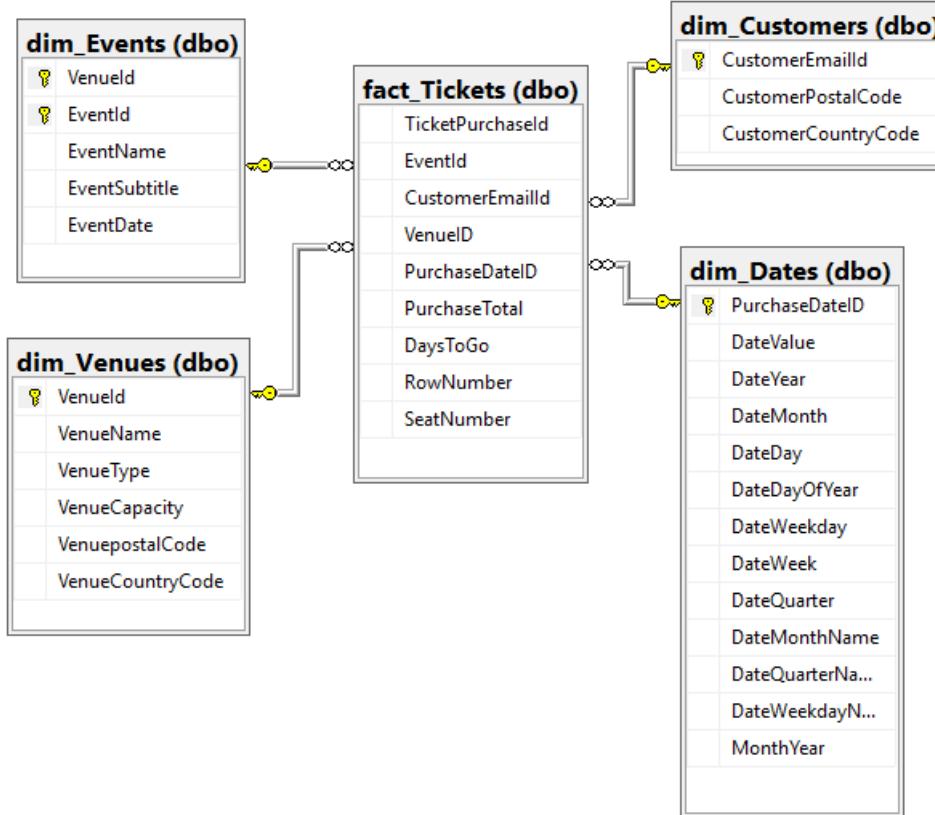
Accessing data for all tenants is simple when all the data is in just one multi-tenant database. But the access is more complex when distributed at scale across potentially thousands of databases. One way to tame the complexity and to minimize the impact of analytics queries on transactional data is to extract data into a purpose designed analytics database or data warehouse.

This tutorial presents a complete analytics scenario for Wingtip Tickets SaaS application. First, *Elastic Jobs* is used to extract data from each tenant database and load it into staging tables in an analytics store. The analytics store could either be an SQL Database or a SQL Data Warehouse. For large-scale data extraction, [Azure Data Factory](#) is recommended.

Next, the aggregated data is transformed into a set of [star-schema](#) tables. The tables consist of a central fact table plus related dimension tables. For Wingtip Tickets:

- The central fact table in the star-schema contains ticket data.
- The dimension tables describe venues, events, customers, and purchase dates.

Together the central fact and dimension tables enable efficient analytical processing. The star-schema used in this tutorial is shown in the following image:



Finally, the analytics store is queried using [PowerBI](#) to highlight insights into tenant behavior and their use of the Wingtip Tickets application. You run queries that:

- Show the relative popularity of each venue
- Highlight patterns in ticket sales for different events
- Show the relative success of different venues in selling out their event

Understanding how each tenant is using the service is used to explore options for monetizing the service and improving the service to help tenants be more successful. This tutorial provides basic examples of the kinds of insights that can be gleaned from tenant data.

## Setup

### Prerequisites

To complete this tutorial, make sure the following prerequisites are met:

- The Wingtip Tickets SaaS Database Per Tenant application is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip SaaS application](#)
- The Wingtip Tickets SaaS Database Per Tenant scripts and application [source code](#) are downloaded from

GitHub. See download instructions. Be sure to *unblock the zip file* before extracting its contents. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

- Power BI Desktop is installed. [Download Power BI Desktop](#)
- The batch of additional tenants has been provisioned, see the [Provision tenants tutorial](#).
- A job account and job account database have been created. See the appropriate steps in the [Schema management tutorial](#).

### Create data for the demo

In this tutorial, analysis is performed on ticket sales data. In the current step, you generate ticket data for all the tenants. Later this data is extracted for analysis. *Ensure you have provisioned the batch of tenants as described earlier, so that you have a meaningful amount of data.* A sufficiently large amount of data can expose a range of different ticket purchasing patterns.

1. In PowerShell ISE, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\Demo-TenantAnalytics.ps1, and set the following value:
  - **\$DemoScenario = 1** Purchase tickets for events at all venues
2. Press **F5** to run the script and create ticket purchasing history for every event in each venue. The script runs for several minutes to generate tens of thousands of tickets.

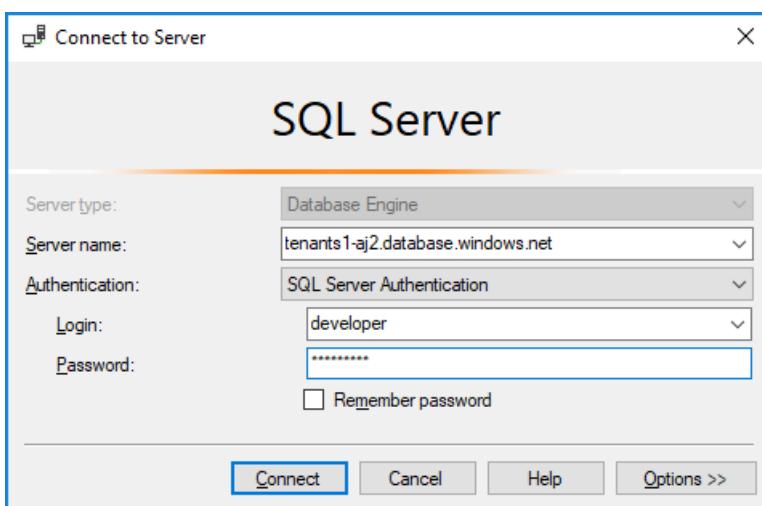
### Deploy the analytics store

Often there are numerous transactional databases that together hold all tenant data. You must aggregate the tenant data from the many transactional databases into one analytics store. The aggregation enables efficient query of the data. In this tutorial, an Azure SQL Database database is used to store the aggregated data.

In the following steps, you deploy the analytics store, which is called **tenantanalytics**. You also deploy predefined tables that are populated later in the tutorial:

1. In PowerShell ISE, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\Demo-TenantAnalytics.ps1
2. Set the \$DemoScenario variable in the script to match your choice of analytics store:
  - To use SQL database without column store, set **\$DemoScenario = 2**
  - To use SQL database with column store, set **\$DemoScenario = 3**
3. Press **F5** to run the demo script (that calls the Deploy-TenantAnalytics<XX>.ps1 script) which creates the tenant analytics store.

Now that you have deployed the application and filled it with interesting tenant data, use [SQL Server Management Studio \(SSMS\)](#) to connect **tenants1-dpt-<User>** and **catalog-dpt-<User>** servers using Login = *developer*, Password = *P@ssword1*. See the [introductory tutorial](#) for more guidance.

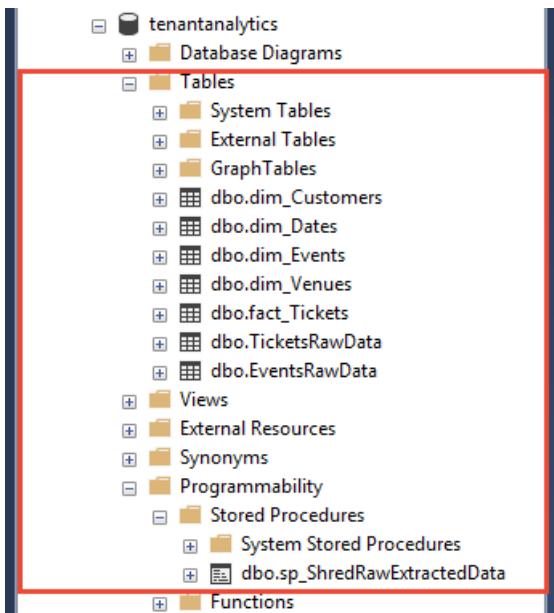


In the Object Explorer, perform the following steps:

1. Expand the *tenants1-dpt-<User>* server.
2. Expand the Databases node, and see the list of tenant databases.
3. Expand the *catalog-dpt-<User>* server.
4. Verify that you see the analytics store and the jobaccount database.

See the following database items in the SSMS Object Explorer by expanding the analytics store node:

- Tables **TicketsRawData** and **EventsRawData** hold raw extracted data from the tenant databases.
- The star-schema tables are **fact\_Tickets**, **dim\_Customers**, **dim\_Venues**, **dim\_Events**, and **dim\_Dates**.
- The stored procedure is used to populate the star-schema tables from the raw data tables.



## Data extraction

### Create target groups

Before proceeding, ensure you have deployed the job account and jobaccount database. In the next set of steps, Elastic Jobs is used to extract data from each tenant database, and to store the data in the analytics store. Then the second job shreds the data and stores it into tables in the star-schema. These two jobs run against two different target groups, namely **TenantGroup** and **AnalyticsGroup**. The extract job runs against the TenantGroup, which contains all the tenant databases. The shredding job runs against the AnalyticsGroup, which contains just the analytics store. Create the target groups by using the following steps:

1. In SSMS, connect to the **jobaccount** database in *catalog-dpt-<User>*.
2. In SSMS, open ... \Learning Modules\Operational Analytics\Tenant Analytics\ TargetGroups.sql
3. Modify the @User variable at the top of the script, replacing <User> with the user value used when you deployed the Wingtip SaaS app.
4. Press **F5** to run the script that creates the two target groups.

### Extract raw data from all tenants

Extensive data modifications might occur more frequently for *ticket* and *customer* data than for *event* and *venue* data. Therefore, consider extracting ticket and customer data separately and more frequently than you extract event and venue data. In this section, you define and schedule two separate jobs:

- Extract ticket and customer data.
- Extract event and venue data.

Each job extracts its data, and posts it into the analytics store. There a separate job shreds the extracted data into the analytics star-schema.

1. In SSMS, connect to the **jobaccount** database in catalog-dpt-<User> server.
2. In SSMS, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\ExtractTickets.sql.
3. Modify @User at the top of the script, and replace <user> with the user name used when you deployed the Wingtip SaaS app
4. Press F5 to run the script that creates and runs the job that extracts tickets and customers data from each tenant database. The job saves the data into the analytics store.
5. Query the TicketsRawData table in the tenantanalytics database, to ensure that the table is populated with tickets information from all tenants.

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The Object Explorer on the left shows the database structure, including the tenantanalytics database which contains the TicketsRawData table. The central pane displays the SQL script 'ExtractTickets.sql'. The results pane below shows a grid of 15 rows of data extracted from the TicketsRawData table, with the last row highlighted. The status bar at the bottom indicates the query was executed successfully.

job_execution_id	job_name	job_id	job_version_number	step_id	is_active	lifecycle	create_time
1	B6FEF7C9-E0104BEE6-AAC5-B201BED399A7	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
2	B6FEF7C9-E0104BEE6-AAC5-B201BED399A7	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
3	4D893CC9-BF6E-4A20-923F-1764400A6A82	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
4	D654DA06-2CE0-400A-B55D-FB10EEDCF333	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
5	0C85841E-2AA6-4FC2-A757-C9CB18CE0F	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
6	0C85841E-2AA6-4FC2-A757-C9CB18CE0F	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
7	0C85841E-2AA6-4FC2-A757-C9CB18CE0F	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
8	4D893CC9-BF6E-4A20-923F-1764400A6A82	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	NULL	0	Succeeded
9	0C85841E-2AA6-4FC2-A757-C9CB18CE0F	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
10	4D893CC9-BF6E-4A20-923F-1764400A6A82	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
11	D654DA06-2CE0-400A-B55D-FB10EEDCF333	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
12	4D893CC9-BF6E-4A20-923F-1764400A6A82	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
13	B6FEF7C9-E0104BEE6-AAC5-B201BED399A7	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
14	B6FEF7C9-E0104BEE6-AAC5-B201BED399A7	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded
15	0C85841E-2AA6-4FC2-A757-C9CB18CE0F	ExtractTickets	0050DCAB-27C5-4368-AB3F-D37A58D82473	1	1	0	Succeeded

Repeat the preceding steps, except this time replace \ExtractTickets.sql with \ExtractVenuesEvents.sql in step 2.

Successfully running the job populates the EventsRawData table in the analytics store with new events and venues information from all tenants.

## Data reorganization

### Shred extracted data to populate star-schema tables

The next step is to shred the extracted raw data into a set of tables that are optimized for analytics queries. A star-schema is used. A central fact table holds individual ticket sales records. Other tables are populated with related data about venues, events, and customers. And there are time dimension tables.

In this section of the tutorial, you define and run a job that merges the extracted raw data with the data in the star-schema tables. After the merge job is finished, the raw data is deleted, leaving the tables ready to be populated by the next tenant data extract job.

1. In SSMS, connect to the **jobaccount** database in catalog-dpt-<User>.
2. In SSMS, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\ShredRawExtractedData.sql.
3. Press F5 to run the script to define a job that calls the sp\_ShredRawExtractedData stored procedure in the analytics store.

4. Allow enough time for the job to run successfully.

- Check the **Lifecycle** column of jobs.jobs\_execution table for the status of job. Ensure that the job **Succeeded** before proceeding. A successful run displays data similar to the following chart:

The screenshot shows the Microsoft SQL Server Management Studio interface. In the Object Explorer, several databases are listed under 'catalog-aj7.database.windows.net (SQL Server)'. In the center pane, a query window titled 'ShredRawExtractedData.sql' is open, displaying T-SQL code to check job execution status. Below the code, the 'Results' tab shows a table with three rows of data from the 'jobs.jobs\_execution' table. The columns are: job\_name, job\_id, job\_version\_number, step\_id, is\_active, lifecycle, and create\_time. The data is as follows:

	job_name	job_id	job_version_number	step_id	is_active	lifecycle	create_time	
1	AC0A07E9	ShredRawExtractedData	643E47BB-DA06-4708-8B58-90F6C31C63A2	1	1	0	Succeeded	2017-11-08
2	AC0A07E9	ShredRawExtractedData	643E47BB-DA06-4708-8B58-90F6C31C63A2	1	1	0	Succeeded	2017-11-08
3	AC0A07E9	ShredRawExtractedData	643E47BB-DA06-4708-8B58-90F6C31C63A2	1	NULL	0	Succeeded	2017-11-08

The bottom status bar indicates 'Query executed successfully.' and shows the session details: catalog-aj7.database.windows.net | developer (101) | jobaccount | 00:00:00 | 3 rows.

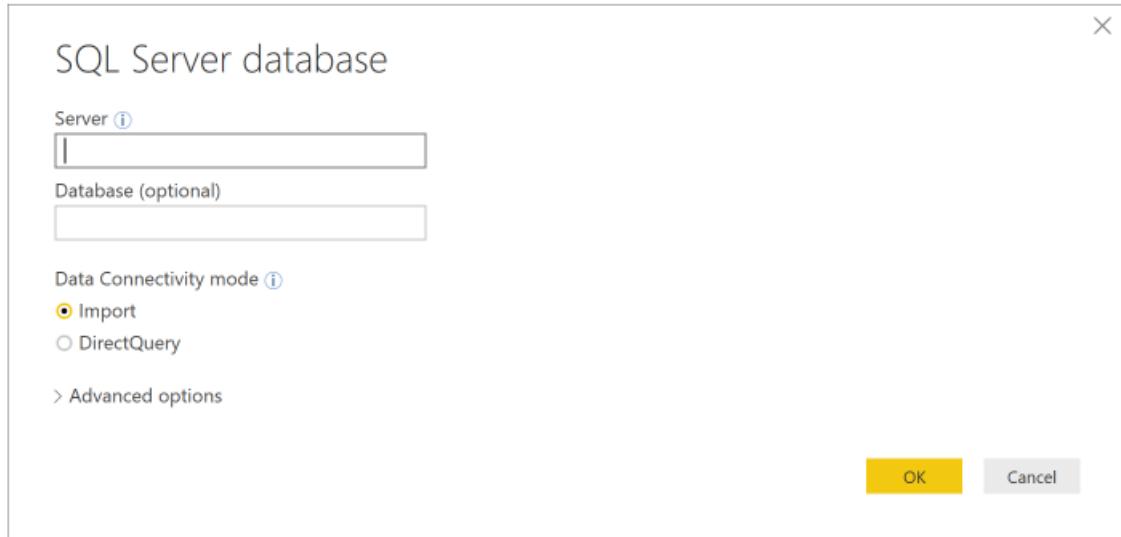
## Data exploration

### Visualize tenant data

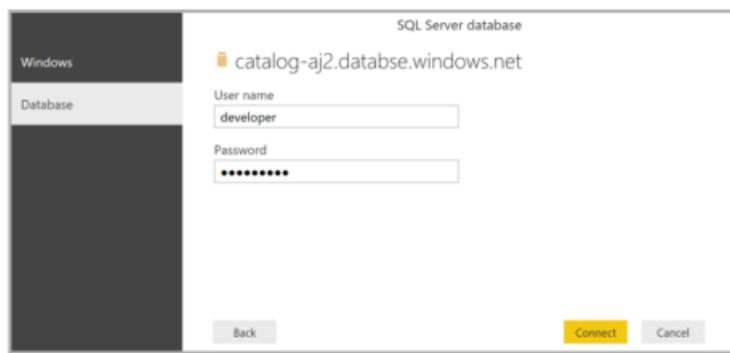
The data in the star-schema table provides all the ticket sales data needed for your analysis. To make it easier to see trends in large data sets, you need to visualize it graphically. In this section, you learn how to use **Power BI** to manipulate and visualize the tenant data you have extracted and organized.

Use the following steps to connect to Power BI, and to import the views you created earlier:

1. Launch Power BI desktop.
2. From the Home ribbon, select **Get Data**, and select **More...** from the menu.
3. In the **Get Data** window, select Azure SQL Database.
4. In the database login window, enter your server name (catalog-dpt-<User>.database.windows.net). Select **Import** for **Data Connectivity Mode**, and then click OK.



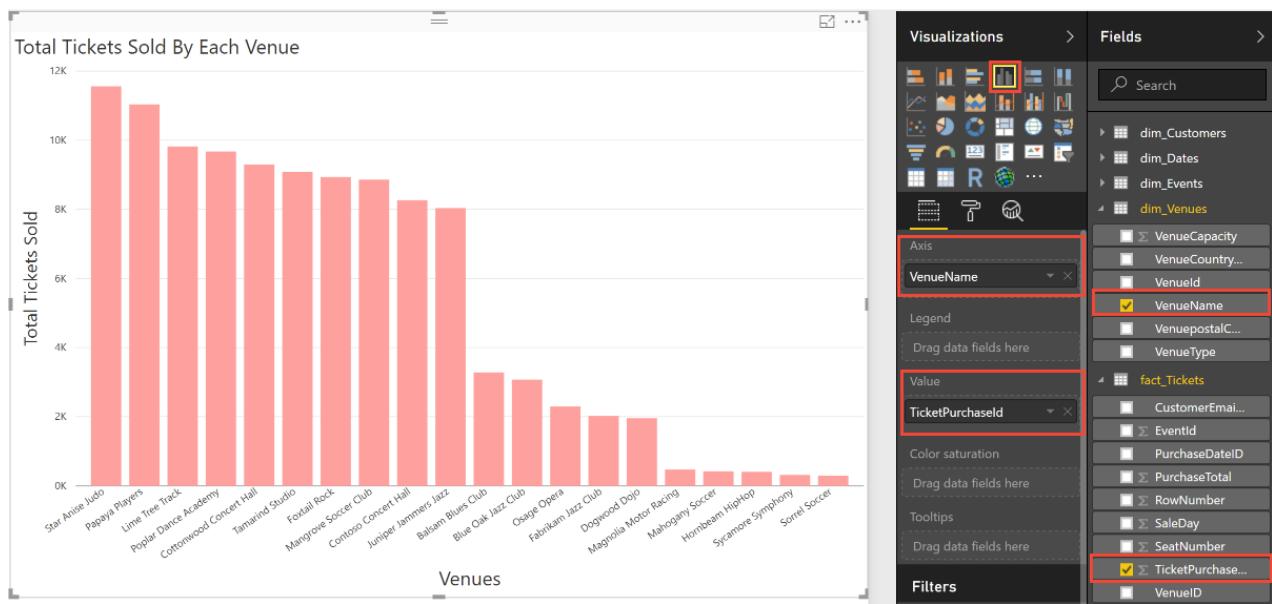
5. Select **Database** in the left pane, then enter user name = *developer*, and enter password = *P@ssword1*. Click **Connect**.



6. In the **Navigator** pane, under the analytics database, select the star-schema tables: fact\_Tickets, dim\_Events, dim\_Venues, dim\_Customers and dim\_Dates. Then select **Load**.

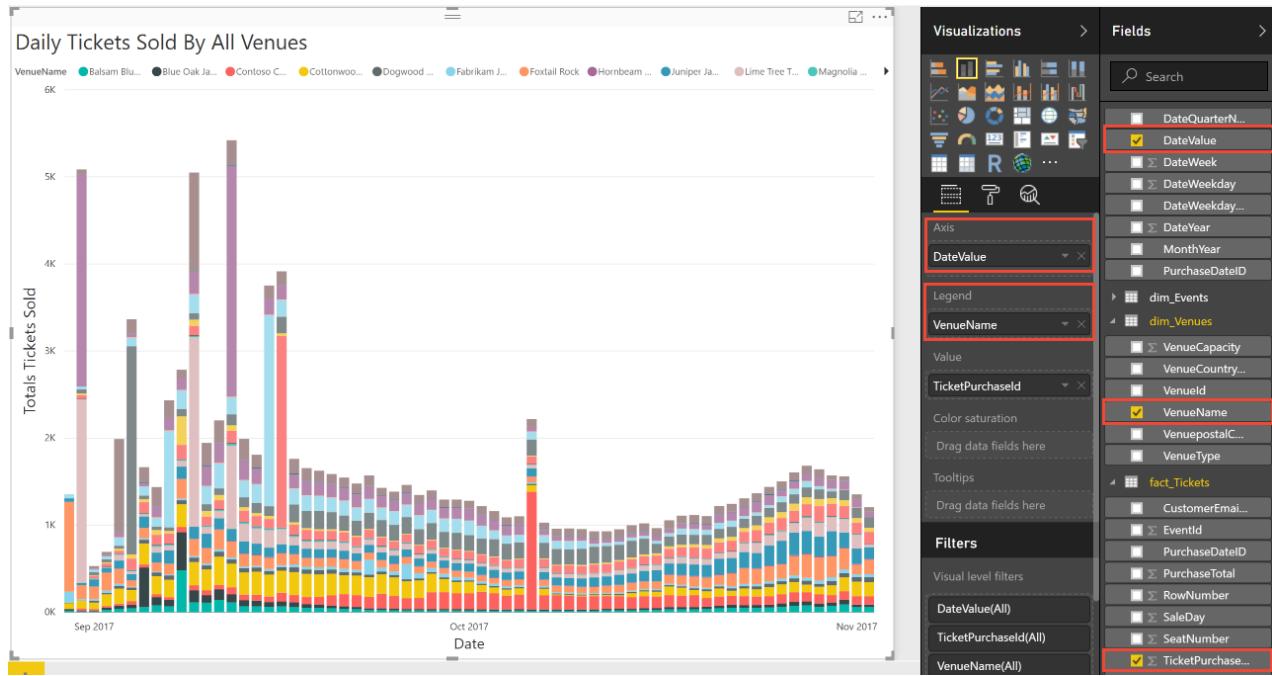
Congratulations! You have successfully loaded the data into Power BI. Now you can start exploring interesting visualizations to help gain insights into your tenants. Next you walk through how analytics can enable you to provide data-driven recommendations to the Wingtip Tickets business team. The recommendations can help to optimize the business model and customer experience.

You start by analyzing ticket sales data to see the variation in usage across the venues. Select the following options in Power BI to plot a bar chart of the total number of tickets sold by each venue. Due to random variation in the ticket generator, your results may be different.



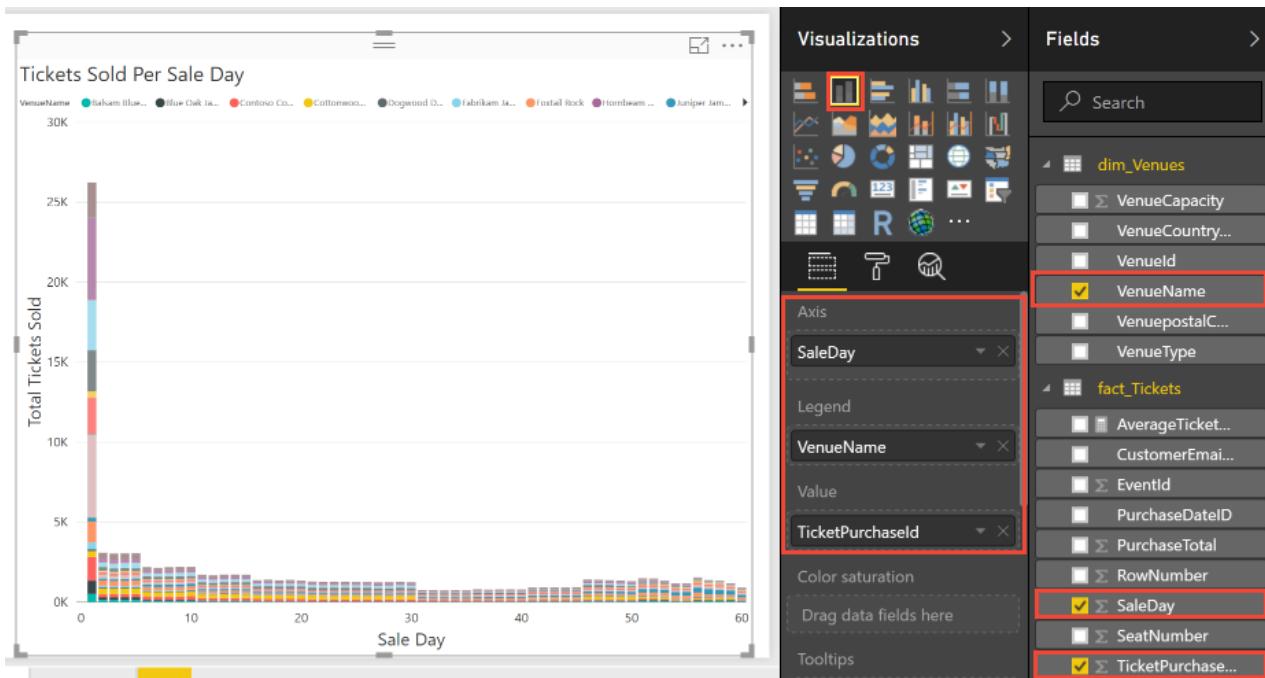
The preceding plot confirms that the number of tickets sold by each venue varies. Venues that sell more tickets are using your service more heavily than venues that sell fewer tickets. There may be an opportunity here to tailor resource allocation according to different tenant needs.

You can further analyze the data to see how ticket sales vary over time. Select the following options in Power BI to plot the total number of tickets sold each day for a period of 60 days.



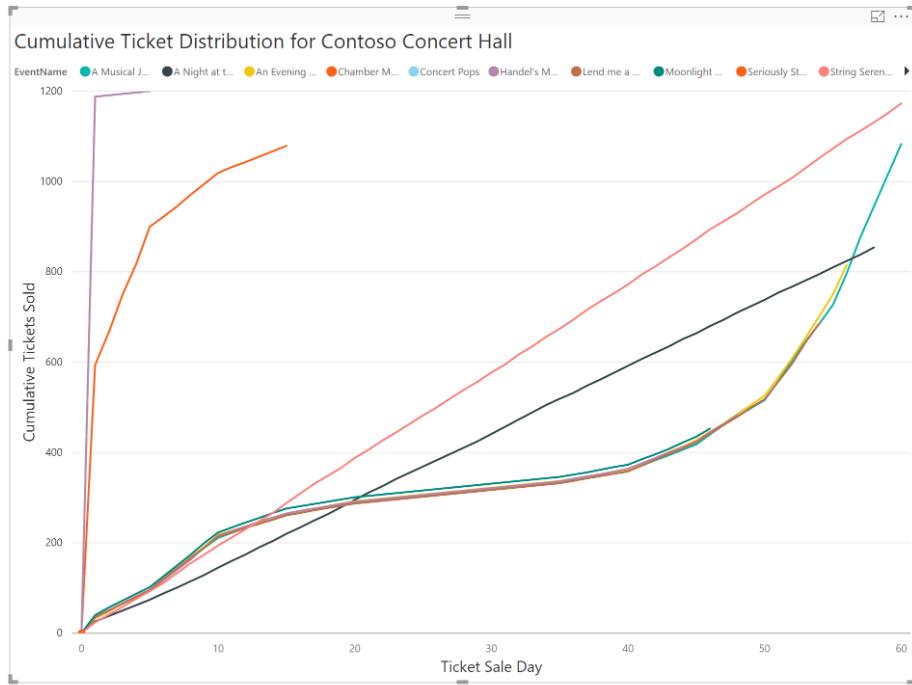
The preceding chart displays that ticket sales spike for some venues. These spikes reinforce the idea that some venues might be consuming system resources disproportionately. So far there is no obvious pattern in when the spikes occur.

Next you want to further investigate the significance of these peak sale days. When do these peaks occur after tickets go on sale? To plot tickets sold per day, select the following options in Power BI.



The preceding plot shows that some venues sell a lot of tickets on the first day of sale. As soon as tickets go on sale at these venues, there seems to be a mad rush. This burst of activity by a few venues might impact the service for other tenants.

You can drill into the data again to see if this mad rush is true for all events hosted by these venues. In previous plots, you observed that Contoso Concert Hall sells a lot of tickets, and that Contoso also has a spike in ticket sales on certain days. Play around with Power BI options to plot cumulative ticket sales for Contoso Concert Hall, focusing on sale trends for each of its events. Do all events follow the same sale pattern?



The preceding plot for Contoso Concert Hall shows that the mad rush does not happen for all events. Play around with the filter options to see sale trends for other venues.

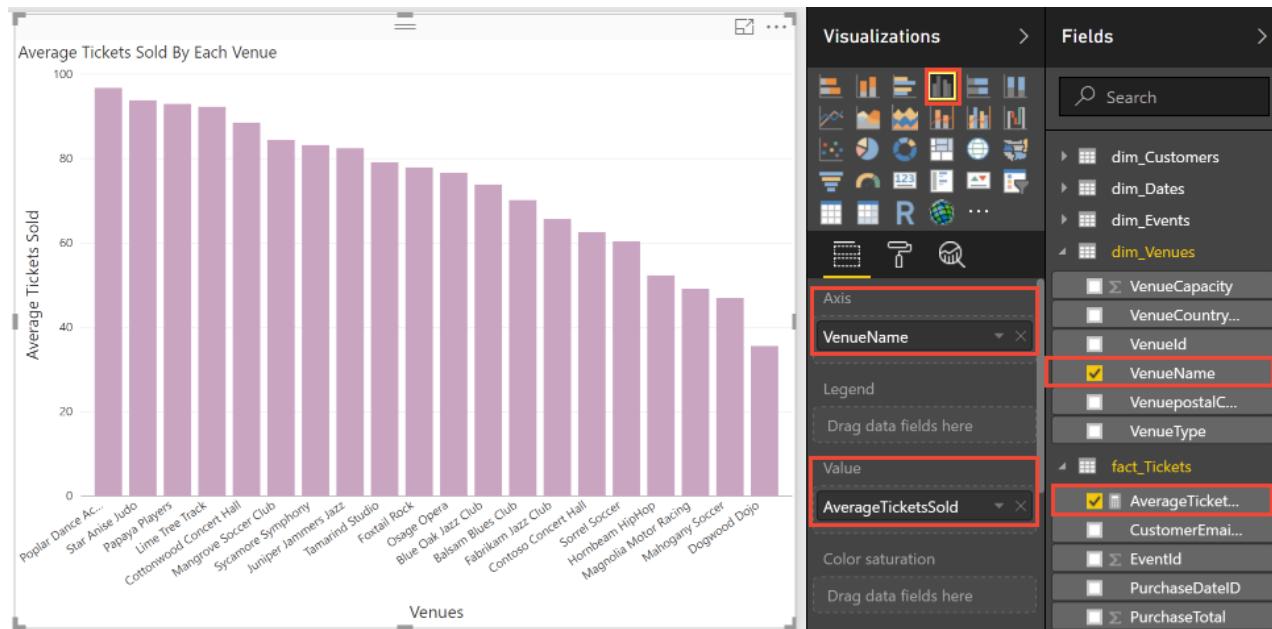
The insights into ticket selling patterns might lead Wingtip Tickets to optimize their business model. Instead of charging all tenants equally, perhaps Wingtip should introduce service tiers with different compute sizes. Larger venues that need to sell more tickets per day could be offered a higher tier with a higher service level agreement (SLA). Those venues could have their databases placed in pool with higher per-database resource limits. Each

service tier could have an hourly sales allocation, with additional fees charged for exceeding the allocation. Larger venues that have periodic bursts of sales would benefit from the higher tiers, and Wingtip Tickets can monetize their service more efficiently.

Meanwhile, some Wingtip Tickets customers complain that they struggle to sell enough tickets to justify the service cost. Perhaps in these insights there is an opportunity to boost ticket sales for underperforming venues. Higher sales would increase the perceived value of the service. Right click fact\_Tickets and select **New measure**. Enter the following expression for the new measure called **AverageTicketsSold**:

```
AverageTicketsSold = AVERAGEX(SUMMARIZE(TableName, TableName[Venue Name]), CALCULATE(SUM(TableName[Tickets Sold])))
```

Select the following visualization options to plot the percentage tickets sold by each venue to determine their relative success.



The preceding plot shows that even though most venues sell more than 80% of their tickets, some are struggling to fill more than half the seats. Play around with the Values Well to select maximum or minimum percentage of tickets sold for each venue.

Earlier you deepened your analysis to discover that ticket sales tend to follow predictable patterns. This discovery might let Wingtip Tickets help underperforming venues boost ticket sales by recommending dynamic pricing. This discovery could reveal an opportunity to employ machine learning techniques to predict ticket sales for each event. Predictions could also be made for the impact on revenue of offering discounts on ticket sales. Power BI Embedded could be integrated into an event management application. The integration could help visualize predicted sales and the effect of different discounts. The application could help devise an optimum discount to be applied directly from the analytics display.

You have observed trends in tenant data from the WingTip application. You can contemplate other ways the app can inform business decisions for SaaS application vendors. Vendors can better cater to the needs of their tenants. Hopefully this tutorial has equipped you with tools necessary to perform analytics on tenant data to empower your businesses to make data-driven decisions.

## Next steps

In this tutorial, you learned how to:

- Deployed a tenant analytics database with pre-defined star schema tables
- Used elastic jobs to extract data from all the tenant database

- Merge the extracted data into tables in a star-schema designed for analytics
- Query an analytics database
- Use Power BI for data visualization to observe trends in tenant data

Congratulations!

## Additional resources

- Additional [tutorials that build upon the Wingtip SaaS application](#).
- [Elastic Jobs](#).
- [Cross-tenant analytics using extracted data - multi-tenant app](#)

# Explore SaaS analytics with Azure SQL Database, SQL Data Warehouse, Data Factory, and Power BI

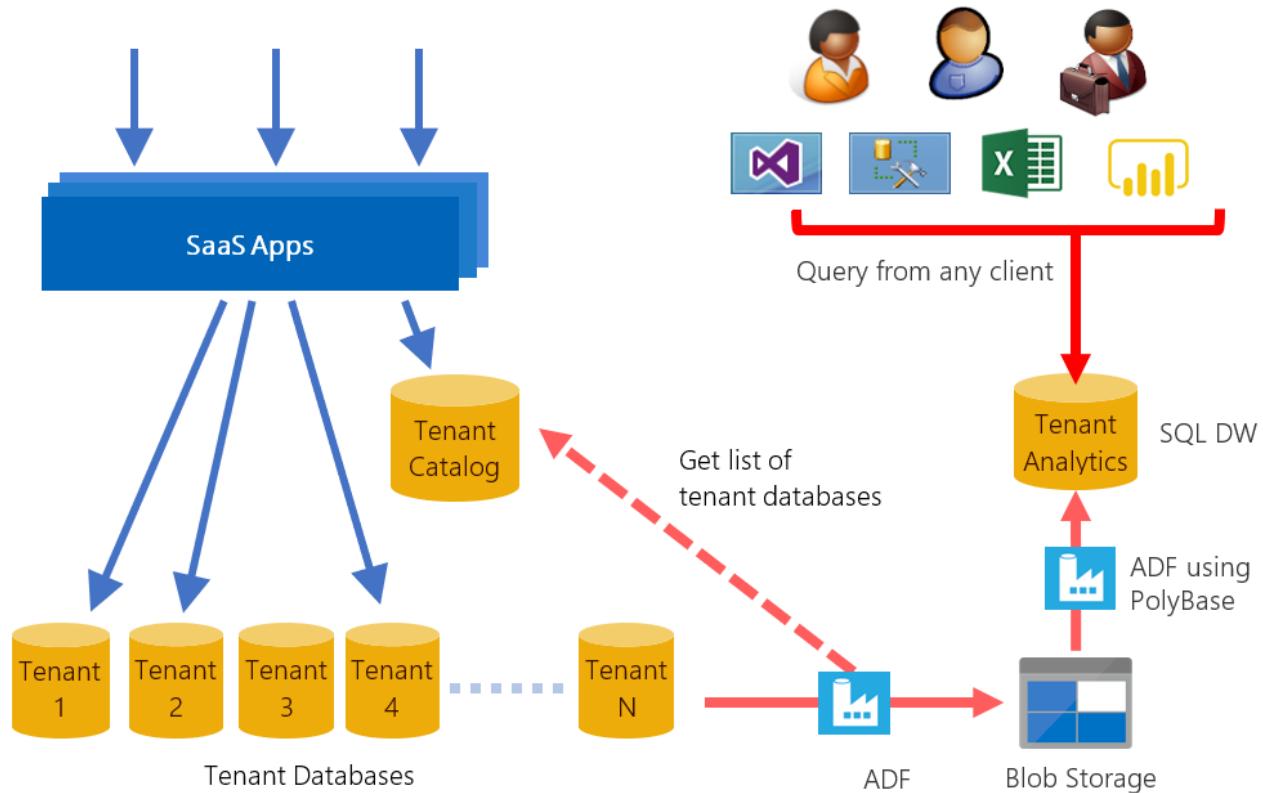
11/7/2019 • 16 minutes to read • [Edit Online](#)

In this tutorial, you walk through an end-to-end analytics scenario. The scenario demonstrates how analytics over tenant data can empower software vendors to make smart decisions. Using data extracted from each tenant database, you use analytics to gain insights into tenant behavior, including their use of the sample Wingtip Tickets SaaS application. This scenario involves three steps:

1. **Extract data** from each tenant database into an analytics store, in this case, a SQL Data Warehouse.
2. **Optimize the extracted data** for analytics processing.
3. Use **Business Intelligence** tools to draw out useful insights, which can guide decision making.

In this tutorial you learn how to:

- Create the tenant analytics store for loading.
- Use Azure Data Factory (ADF) to extract data from each tenant database into the analytics data warehouse.
- Optimize the extracted data (reorganize into a star-schema).
- Query the analytics data warehouse.
- Use Power BI for data visualization to highlight trends in tenant data and make recommendation for improvements.



## Analytics over extracted tenant data

SaaS applications hold a potentially vast amount of tenant data in the cloud. This data can provide a rich source of insights about the operation and usage of your application, and the behavior of your tenants. These insights can guide feature development, usability improvements, and other investments in the apps and platform.

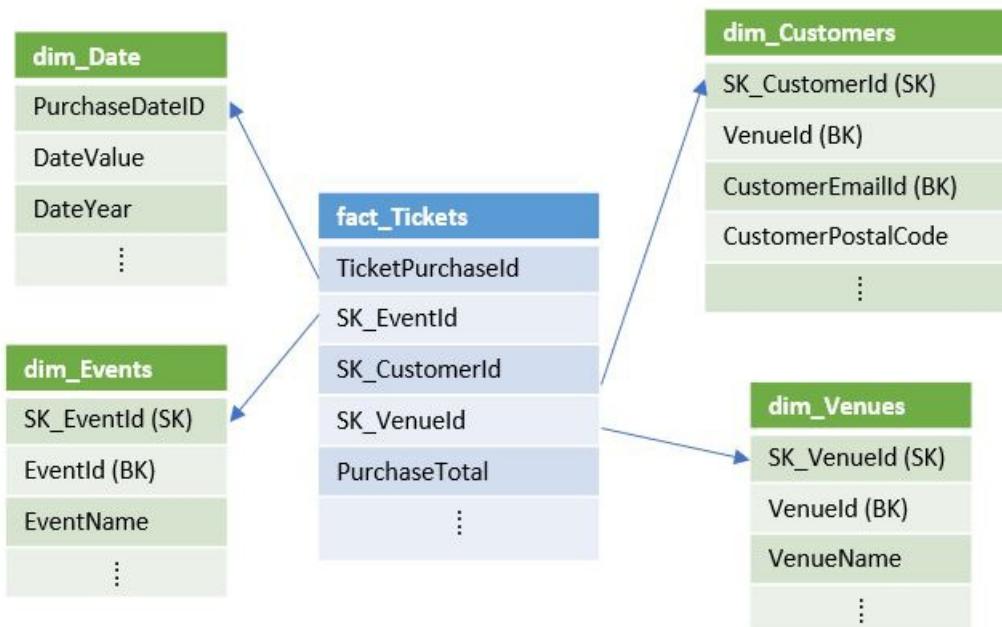
Accessing the data for all tenants is simple when all the data is in just one multi-tenant database. But access is more complex when distributed at scale across thousands of databases. One way to tame the complexity is to extract the data to an analytics database or a data warehouse for query.

This tutorial presents an end-to-end analytics scenario for the Wingtip Tickets application. First, [Azure Data Factory \(ADF\)](#) is used as the orchestration tool to extract tickets sales and related data from each tenant database. This data is loaded into staging tables in an analytics store. The analytics store could either be an SQL Database or a SQL Data Warehouse. This tutorial uses [SQL Data Warehouse](#) as the analytics store.

Next, the extracted data is transformed and loaded into a set of [star-schema](#) tables. The tables consist of a central fact table plus related dimension tables:

- The central fact table in the star-schema contains ticket data.
- The dimension tables contain data about venues, events, customers, and purchase dates.

Together the central and dimension tables enable efficient analytical processing. The star-schema used in this tutorial is displayed in the following image:



Finally, the star-schema tables are queried. Query results are displayed visually using Power BI to highlight insights into tenant behavior and their use of the application. With this star-schema, you run queries that expose:

- Who is buying tickets and from which venue.
- Patterns and trends in the sale of tickets.
- The relative popularity of each venue.

This tutorial provides basic examples of insights that can be gleaned from the Wingtip Tickets data. Understanding how each venue uses the service might cause the Wingtip Tickets vendor to think about different service plans targeted at more or less active venues, for example.

## Setup

### Prerequisites

## NOTE

This tutorial uses features of the Azure Data Factory that are currently in a limited preview (linked service parameterization). If you wish to do this tutorial, provide your subscription ID [here](#). We will send you a confirmation as soon as your subscription has been enabled.

To complete this tutorial, make sure the following prerequisites are met:

- The Wingtip Tickets SaaS Database Per Tenant application is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip SaaS application](#).
- The Wingtip Tickets SaaS Database Per Tenant scripts and application [source code](#) are downloaded from GitHub. See download instructions. Be sure to *unlock the zip file* before extracting its contents.
- Power BI Desktop is installed. [Download Power BI Desktop](#).
- The batch of additional tenants has been provisioned, see the [Provision tenants tutorial](#).

## Create data for the demo

This tutorial explores analytics over ticket sales data. In this step, you generate ticket data for all the tenants. In a later step, this data is extracted for analysis. *Ensure you provisioned the batch of tenants* (as described earlier) so that you have enough data to expose a range of different ticket purchasing patterns.

1. In PowerShell ISE, open ... \Learning Modules\Operational Analytics\Tenant Analytics DW\Demo-TenantAnalyticsDW.ps1, and set the following value:
  - **\$DemoScenario = 1** Purchase tickets for events at all venues
2. Press **F5** to run the script and create ticket purchasing history for all the venues. With 20 tenants, the script generates tens of thousands of tickets and may take 10 minutes or more.

## Deploy SQL Data Warehouse, Data Factory, and Blob Storage

In the Wingtip Tickets app, the tenants' transactional data is distributed over many databases. Azure Data Factory (ADF) is used to orchestrate the Extract, Load, and Transform (ELT) of this data into the data warehouse. To load data into SQL Data Warehouse most efficiently, ADF extracts data into intermediate blob files and then uses [PolyBase](#) to load the data into the data warehouse.

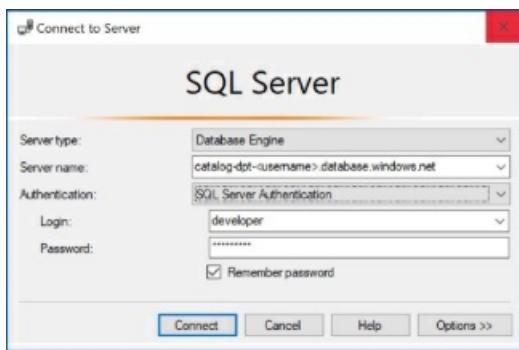
In this step, you deploy the additional resources used in the tutorial: a SQL Data Warehouse called *tenantanalytics*, an Azure Data Factory called *dbtodwload-<user>*, and an Azure storage account called *wingtipstaging-<user>*. The storage account is used to temporarily hold extracted data files as blobs before they are loaded into the data warehouse. This step also deploys the data warehouse schema and defines the ADF pipelines that orchestrate the ELT process.

1. In PowerShell ISE, open ... \Learning Modules\Operational Analytics\Tenant Analytics DW\Demo-TenantAnalyticsDW.ps1 and set:
  - **\$DemoScenario = 2** Deploy tenant analytics data warehouse, blob storage, and data factory
2. Press **F5** to run the demo script and deploy the Azure resources.

Now review the Azure resources you deployed:

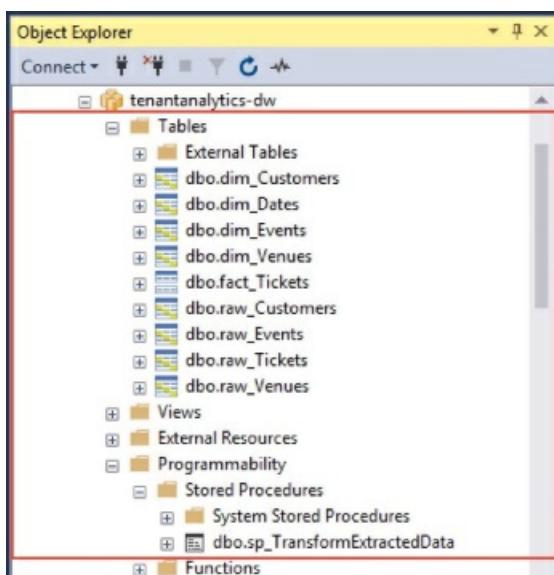
### Tenant databases and analytics store

Use [SQL Server Management Studio \(SSMS\)](#) to connect to **tenants1-dpt-<user>** and **catalog-dpt-<user>** servers. Replace <user> with the value used when you deployed the app. Use Login = *developer* and Password = *P@ssword1*. See the [introductory tutorial](#) for more guidance.



In the Object Explorer:

1. Expand the *tenants1-dpt-<user>* server.
2. Expand the Databases node, and see the list of tenant databases.
3. Expand the *catalog-dpt-<user>* server.
4. Verify that you see the analytics store containing the following objects:
  - a. Tables **raw\_Tickets**, **raw\_Customers**, **raw\_Events** and **raw\_Venues** hold raw extracted data from the tenant databases.
  - b. The star-schema tables are **fact\_Tickets**, **dim\_Customers**, **dim\_Venues**, **dim\_Events**, and **dim\_Dates**.
  - c. The stored procedure, **sp\_transformExtractedData** is used to transform the data and load it into the star-schema tables.



#### Blob storage

1. In the [Azure Portal](#), navigate to the resource group that you used for deploying the application. Verify that a storage account called **wingtipstaging<user>** has been added.

<input type="checkbox"/>	<a href="#">sycamoresymphony (tenants1-dpt-bgd/sycamoresymphony)</a>	SQL database	East US 2
<input type="checkbox"/>	<a href="#">tamarindstudio (tenants1-dpt-bgd/tamarindstudio)</a>	SQL database	East US 2
<input type="checkbox"/>	<a href="#">wingtipstagingbgd</a>	Storage account	East US

2. Click **wingtipstaging<user>** storage account to explore the objects present.
3. Click **Blobs** tile
4. Click the container **configfile**
5. Verify that **configfile** contains a JSON file called **TableConfig.json**. This file contains the source and destination table names, column names, and tracker column name.

## Azure Data Factory (ADF)

In the [Azure Portal](#) in the resource group, verify that an Azure Data Factory called `dbtodownload-<user>` has been added.

	tenantanalytics-dw (catalog-dpt-bgd/tenantanalytics-dw)	SQL data warehouse	East US 2
	tenantcatalog (catalog-dpt-bgd/tenantcatalog)	SQL database	East US 2
	dbtodownload-bgd	Data factory (V2)	West Europe

This section explores the data factory created. Follow the steps below to launch the data factory:

1. In the portal, click the data factory called **dbtodownload-<user>**.
2. Click **Author & Monitor** tile to launch the Data Factory designer in a separate tab.

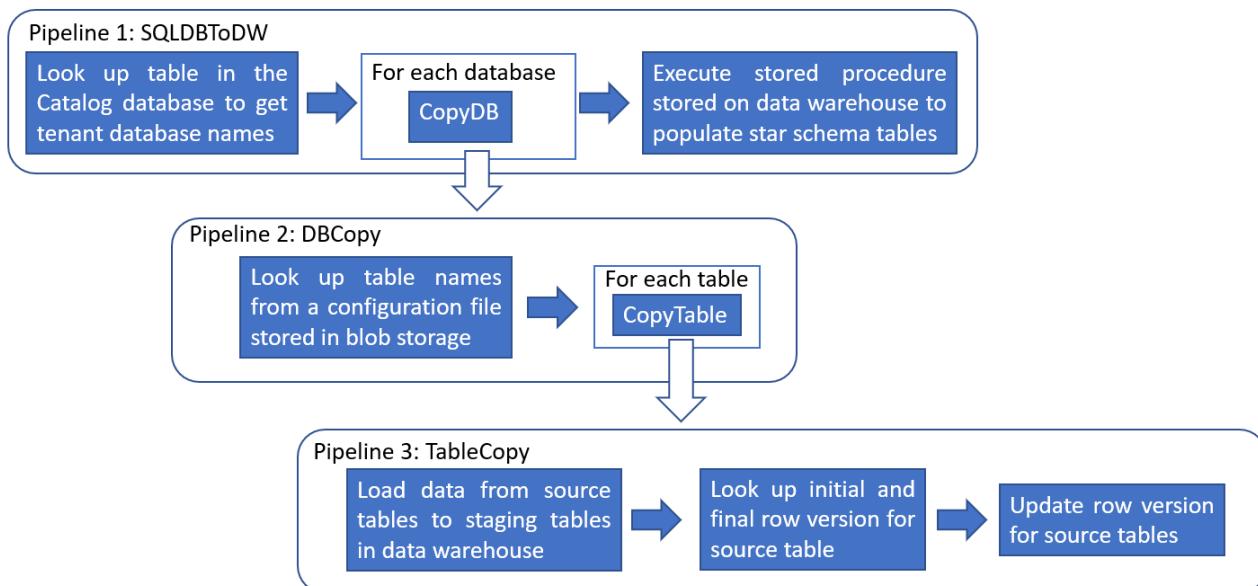
## Extract, Load, and Transform data

Azure Data Factory is used for orchestrating extraction, loading, and transformation of data. In this tutorial, you extract data from four different SQL views from each of the tenant databases: **rawTickets**, **rawCustomers**, **rawEvents**, and **rawVenues**. These views include venue Id, so you can discriminate data from each venue in the data warehouse. The data is loaded into corresponding staging tables in the data warehouse: **raw\_Tickets**, **raw\_customers**, **raw\_Events** and **raw\_Venue**. A stored procedure then transforms the raw data and populates the star-schema tables: **fact\_Tickets**, **dim\_Customers**, **dim\_Venues**, **dim\_Events**, and **dim\_Dates**.

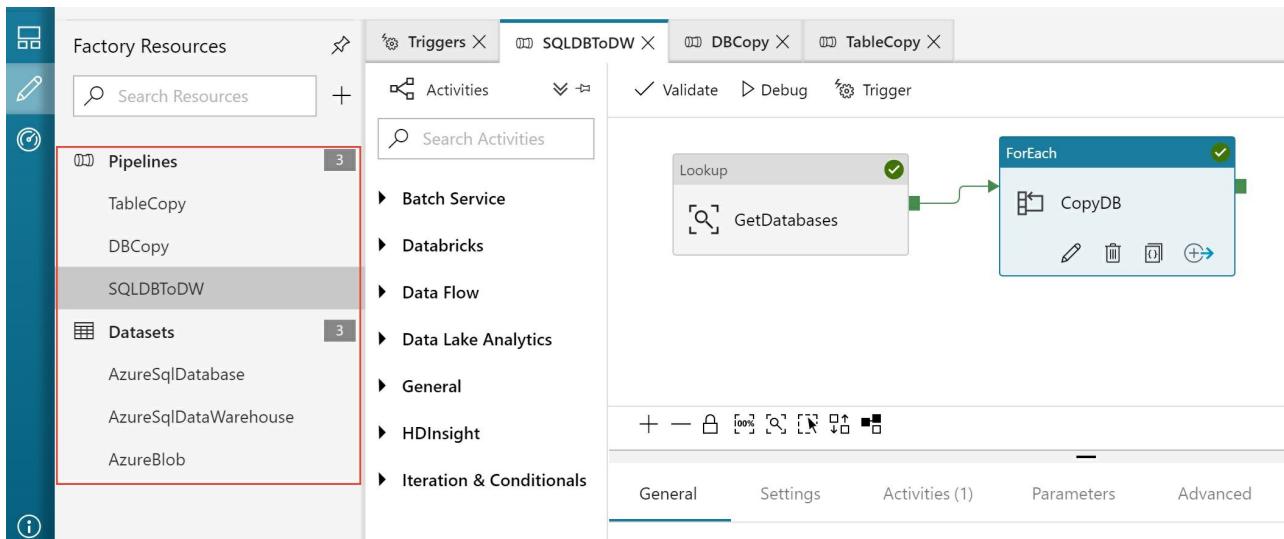
In the previous section, you deployed and initialized the necessary Azure resources, including the data factory. The deployed data factory includes the pipelines, datasets, linked services, etc., required to extract, load, and transform the tenant data. Let's explore these objects further and then trigger the pipeline to move data from tenant databases to the data warehouse.

### Data factory pipeline overview

This section explores the objects created in the data factory. The following figure describes the overall workflow of the ADF pipeline used in this tutorial. If you prefer to explore the pipeline later and see the results first, skip to the next section **Trigger the pipeline run**.



In the overview page, switch to **Author** tab on the left panel and observe that there are three [pipelines](#) and three [datasets](#) created.



The three nested pipelines are: SQLDBToDW, DBCopy, and TableCopy.

**Pipeline 1 - SQLDBToDW** looks up the names of the tenant databases stored in the Catalog database (table name: [\_\_ShardManagement].[ShardsGlobal]) and for each tenant database, executes the **DBCopy** pipeline. Upon completion, the provided **sp\_TransformExtractedData** stored procedure schema, is executed. This stored procedure transforms the loaded data in the staging tables and populates the star-schema tables.

**Pipeline 2 - DBCopy** looks up the names of the source tables and columns from a configuration file stored in blob storage. The **TableCopy** pipeline is then run for each of the four tables: TicketFacts, CustomerFacts, EventFacts, and VenueFacts. The **ForEach** activity executes in parallel for all 20 databases. ADF allows a maximum of 20 loop iterations to be run in parallel. Consider creating multiple pipelines for more databases.

**Pipeline 3 - TableCopy** uses row version numbers in SQL Database (*rowversion*) to identify rows that have been changed or updated. This activity looks up the start and the end row version for extracting rows from the source tables. The **CopyTracker** table stored in each tenant database tracks the last row extracted from each source table in each run. New or changed rows are copied to the corresponding staging tables in the data warehouse: **raw\_Tickets**, **raw\_Customers**, **raw\_Venues**, and **raw\_Events**. Finally the last row version is saved in the **CopyTracker** table to be used as the initial row version for the next extraction.

There are also three parameterized linked services that link the data factory to the source SQL Databases, the target SQL Data Warehouse, and the intermediate Blob storage. In the **Author** tab, click on **Connections** to explore the linked services, as shown in the following image:

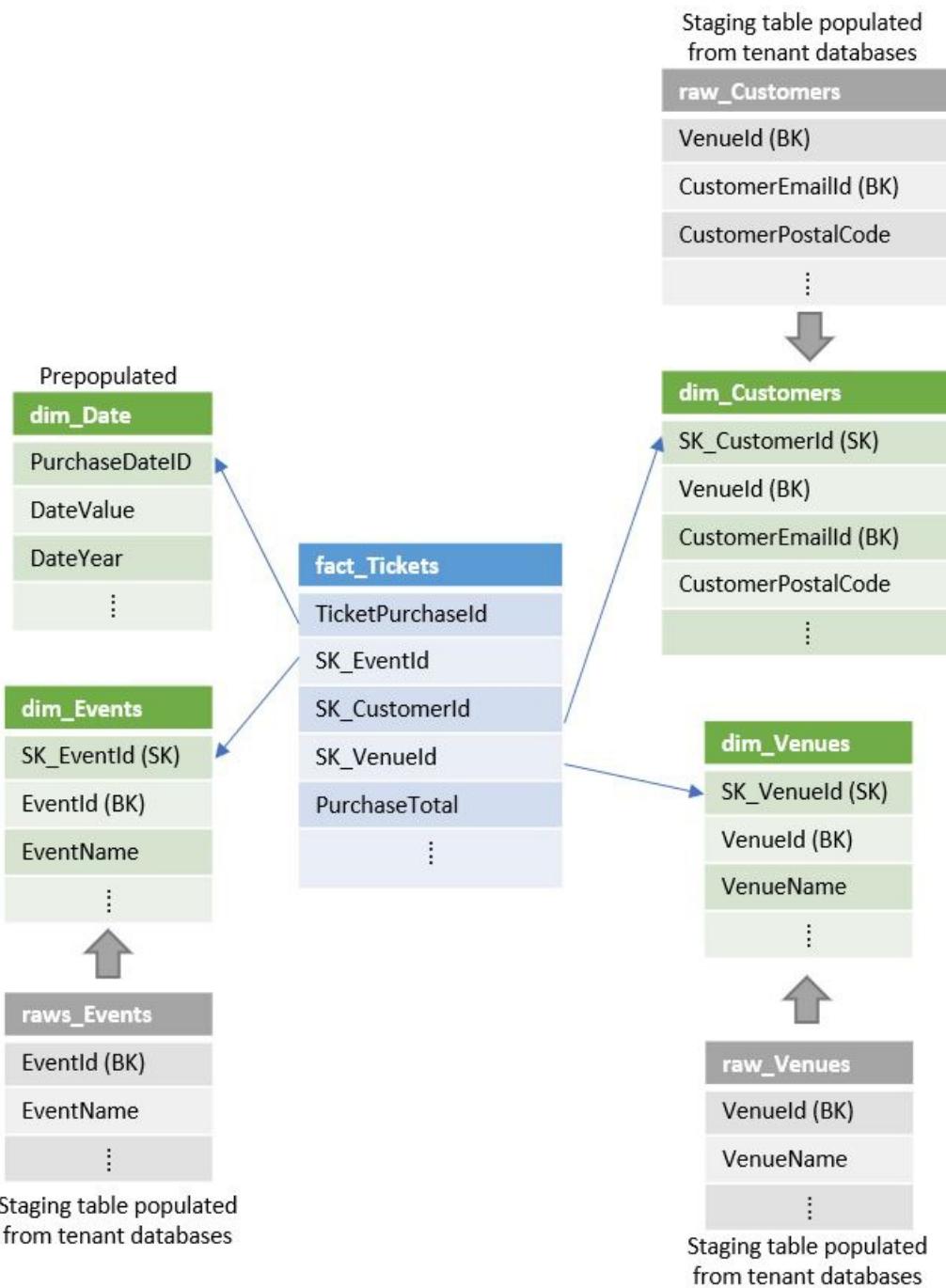
The screenshot shows the 'Connections' blade in the Azure Data Factory interface. On the left sidebar, there are icons for Home, Create, Pipelines, Datasets, and Connections. The 'Connections' icon is highlighted. The main area displays a table of three linked services:

Name	Actions	Type
AzureSqlDatabase	Edit, Delete, Open	Azure SQL Database
AzureSqlDataWarehouse	Edit, Delete, Open	Azure SQL Data Warehouse
AzureStorage	Edit, Delete, Open	Azure Storage

Corresponding to the three linked services, there are three datasets that refer to the data you use in the pipeline activities as inputs or outputs. Explore each of the datasets to observe connections and parameters used. *AzureBlob* points to the configuration file containing source and target tables and columns, as well as the tracker column in each source.

### Data warehouse pattern overview

SQL Data Warehouse is used as the analytics store to perform aggregation on the tenant data. In this sample, PolyBase is used to load data into the SQL Data warehouse. Raw data is loaded into staging tables that have an identity column to keep track of rows that have been transformed into the star-schema tables. The following image shows the loading pattern:



Slowly Changing Dimension (SCD) type 1 dimension tables are used in this example. Each dimension has a surrogate key defined using an identity column. As a best practice, the date dimension table is pre-populated to save time. For the other dimension tables, a CREATE TABLE AS SELECT... (CTAS) statement is used to create a temporary table containing the existing modified and non-modified rows, along with the surrogate keys. This is done with IDENTITY\_INSERT=ON. New rows are then inserted into the table with IDENTITY\_INSERT=OFF. For easy roll-back, the existing dimension table is renamed and the temporary table is renamed to become the new dimension table. Before each run, the old dimension table is deleted.

Dimension tables are loaded before the fact table. This sequencing ensures that for each arriving fact, all referenced dimensions already exist. As the facts are loaded, the business key for each corresponding dimension is matched and the corresponding surrogate keys are added to each fact.

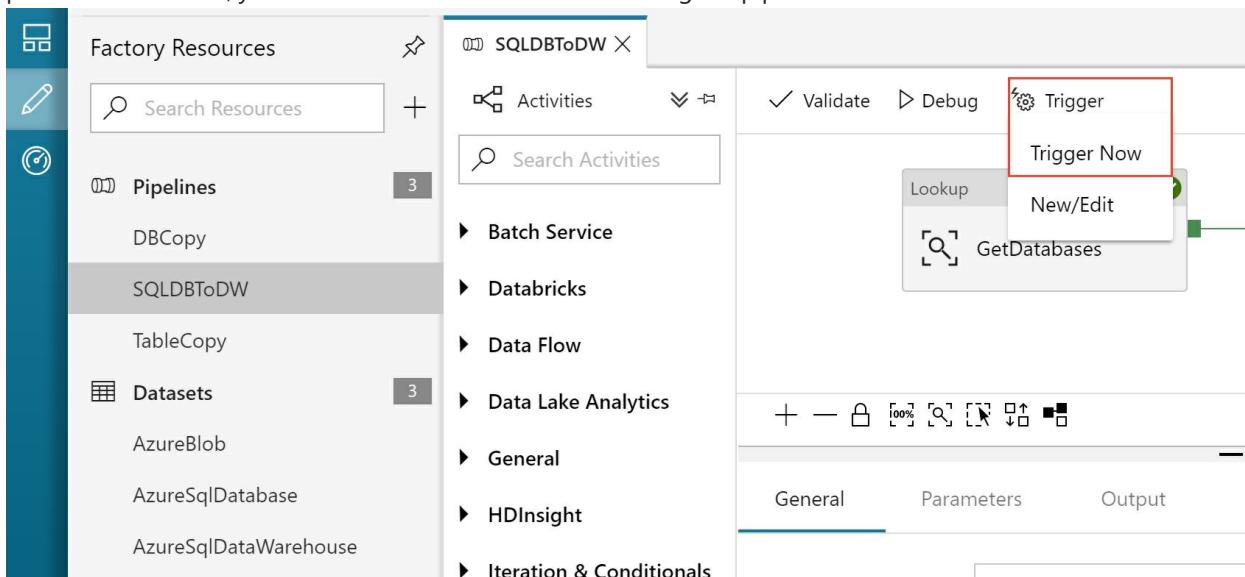
The final step of the transform deletes the staging data ready for the next execution of the pipeline.

### Trigger the pipeline run

Follow the steps below to run the complete extract, load, and transform pipeline for all the tenant databases:

1. In the **Author** tab of the ADF user interface, select **SQLDBToDW** pipeline from the left pane.

2. Click **Trigger** and from the pulled down menu click **Trigger Now**. This action runs the pipeline immediately. In a production scenario, you would define a timetable for running the pipeline to refresh the data on a schedule.



3. On **Pipeline Run** page, click **Finish**.

### Monitor the pipeline run

1. In the ADF user interface, switch to the **Monitor** tab from the menu on the left.
2. Click **Refresh** until SQLDBToDW pipeline's status is **Succeeded**.

Pipeline Name	Actions	Run Start	Duration	Triggered By	Status	Parameters
DBCopy	↻ ▶	03/19/2018, 3:04:59 PM	00:01:55	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
TableCopy	↻ ▶	03/19/2018, 3:03:32 PM	00:01:15	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
TableCopy	↻ ▶	03/19/2018, 3:03:32 PM	00:01:17	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
TableCopy	↻ ▶	03/19/2018, 3:03:32 PM	00:01:20	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
DBCopy	↻ ▶	03/19/2018, 3:03:05 PM	00:01:52	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
TableCopy	↻ ▶	03/19/2018, 3:01:27 PM	00:01:27	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
TableCopy	↻ ▶	03/19/2018, 3:01:27 PM	00:01:29	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
TableCopy	↻ ▶	03/19/2018, 3:01:27 PM	00:01:22	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
DBCopy	↻ ▶	03/19/2018, 3:00:58 PM	00:02:03	PipelineActivity	<span style="color: green;">✓ Succeeded</span>	[@]
SQLDBToDW	↻ ▶	03/19/2018, 3:00:34 PM	00:40:45	Manual trigger	<span style="color: green;">✓ Succeeded</span>	

3. Connect to the data warehouse with SSMS and query the star-schema tables to verify that data was loaded in these tables.

Once the pipeline has completed, the fact table holds ticket sales data for all venues and the dimension tables are populated with the corresponding venues, events, and customers.

## Data Exploration

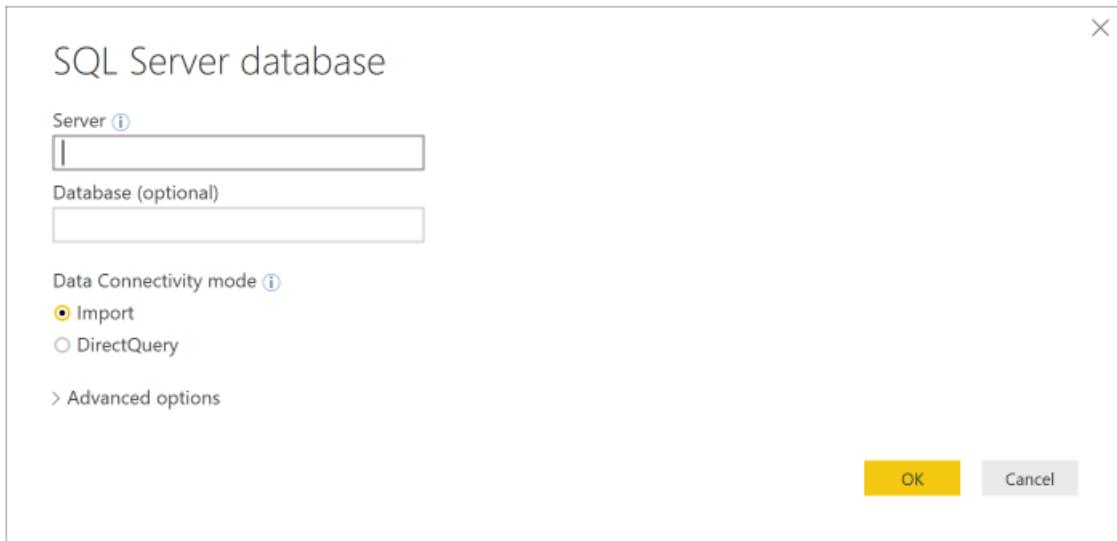
### Visualize tenant data

The data in the star-schema provides all the ticket sales data needed for your analysis. Visualizing data graphically makes it easier to see trends in large data sets. In this section, you use **Power BI** to manipulate and visualize the tenant data in the data warehouse.

Use the following steps to connect to Power BI, and to import the views you created earlier:

1. Launch Power BI desktop.

2. From the Home ribbon, select **Get Data**, and select **More...** from the menu.
3. In the **Get Data** window, select **Azure SQL Database**.
4. In the database login window, enter your server name (**catalog-dpt-<User>.database.windows.net**). Select **Import** for **Data Connectivity Mode**, and then click **OK**.



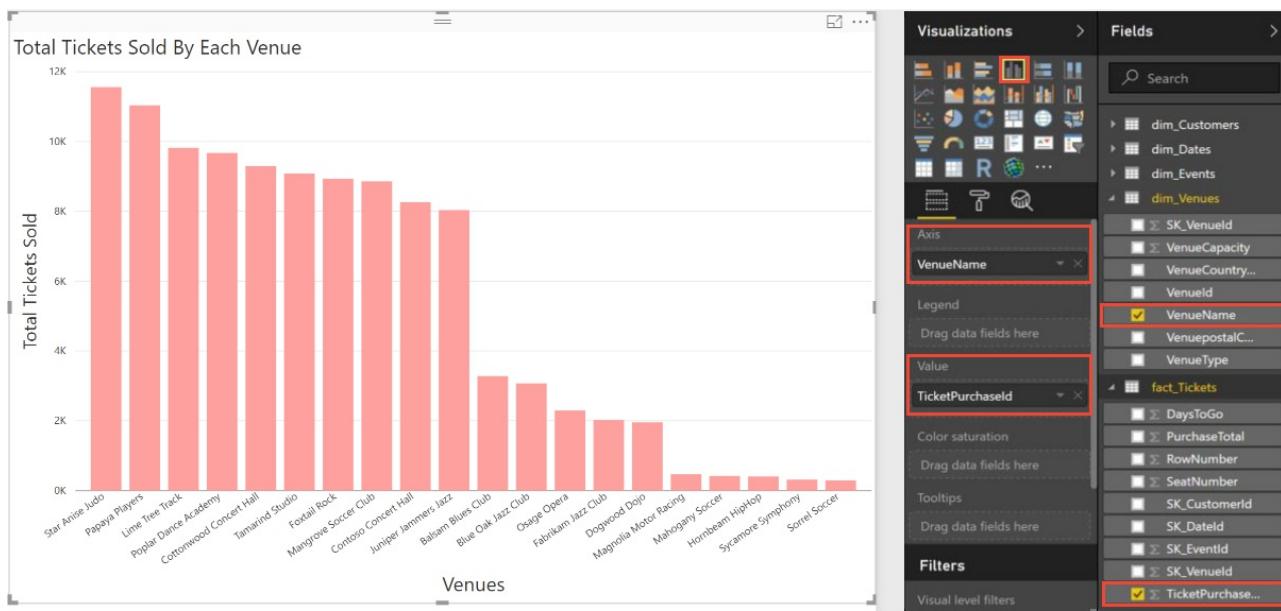
5. Select **Database** in the left pane, then enter user name = *developer*, and enter password = *P@ssword1*. Click **Connect**.



6. In the **Navigator** pane, under the analytics database, select the star-schema tables: **fact\_Tickets**, **dim\_Events**, **dim\_Venues**, **dim\_Customers** and **dim\_Dates**. Then select **Load**.

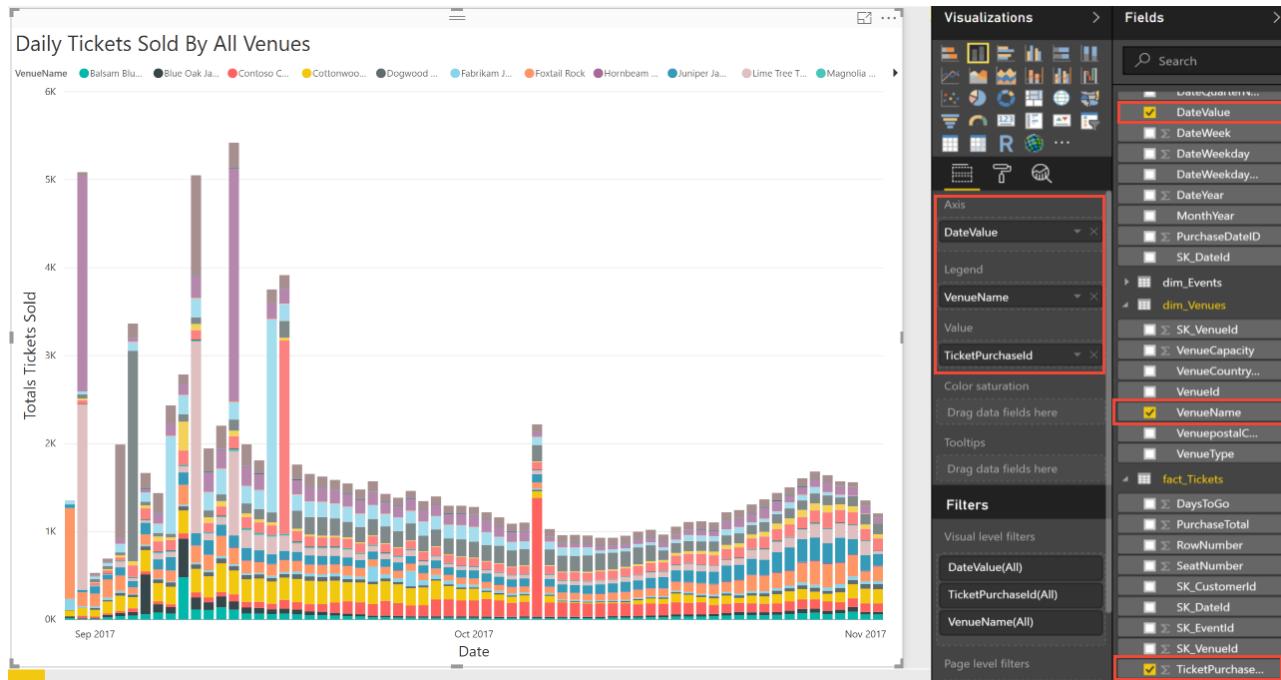
Congratulations! You successfully loaded the data into Power BI. Now explore interesting visualizations to gain insights into your tenants. Let's walk through how analytics can provide some data-driven recommendations to the Wingtip Tickets business team. The recommendations can help to optimize the business model and customer experience.

Start by analyzing ticket sales data to see the variation in usage across the venues. Select the options shown in Power BI to plot a bar chart of the total number of tickets sold by each venue. (Due to random variation in the ticket generator, your results may be different.)



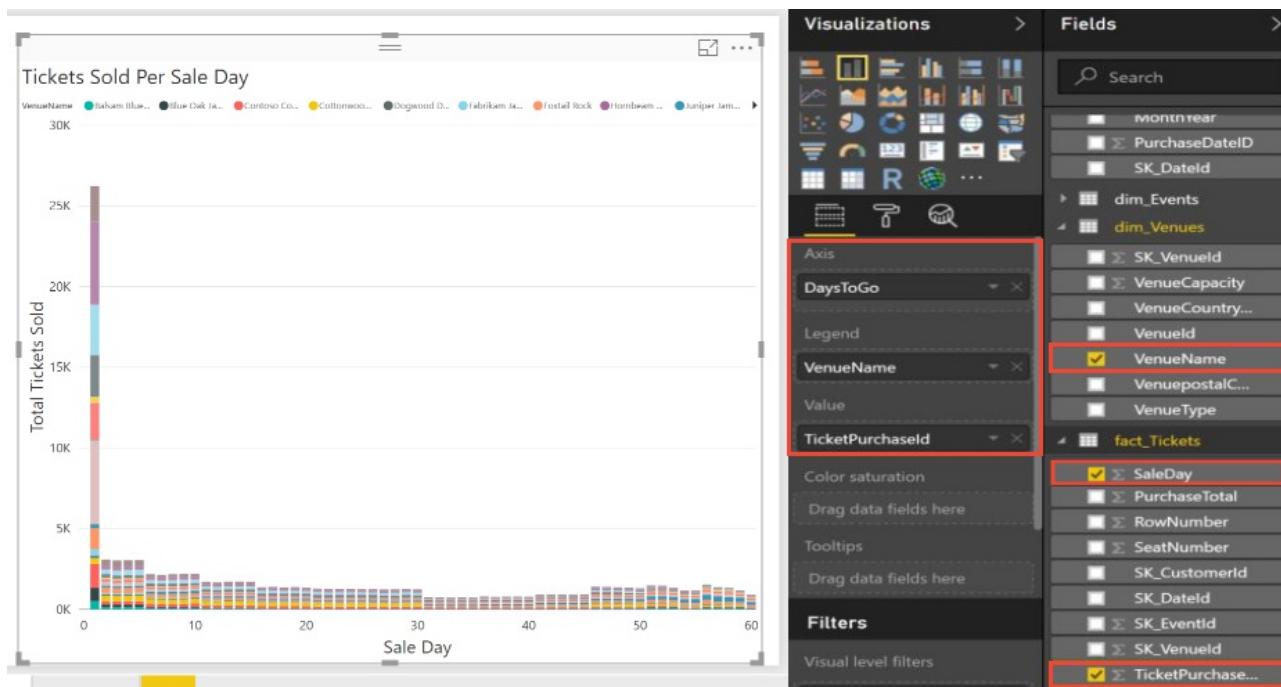
The preceding plot confirms that the number of tickets sold by each venue varies. Venues that sell more tickets are using your service more heavily than venues that sell fewer tickets. There may be an opportunity here to tailor resource allocation according to different tenant needs.

You can further analyze the data to see how ticket sales vary over time. Select the options shown in the following image in Power BI to plot the total number of tickets sold each day for a period of 60 days.



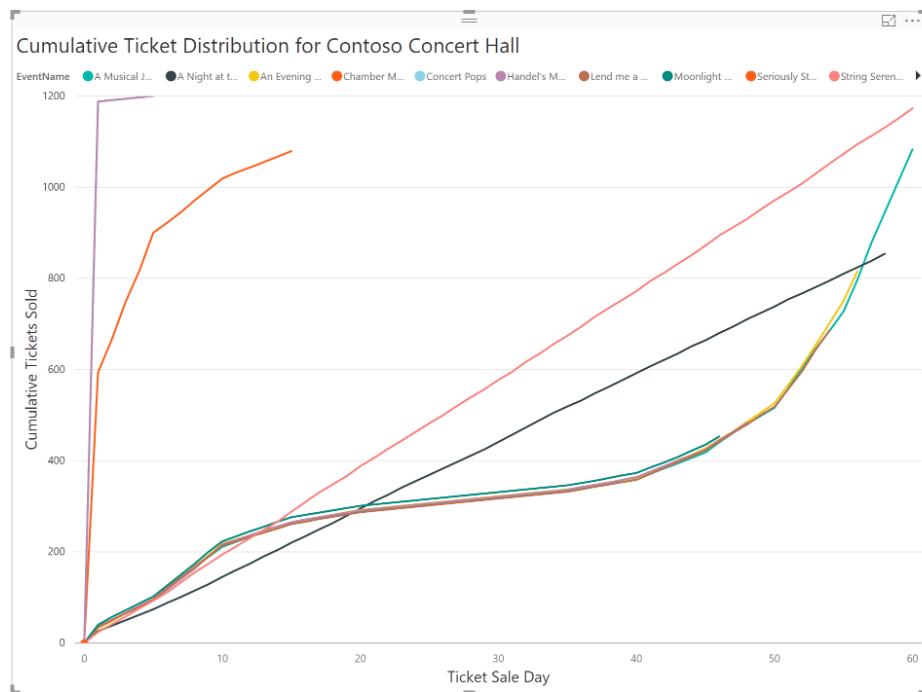
The preceding chart shows that ticket sales spike for some venues. These spikes reinforce the idea that some venues might be consuming system resources disproportionately. So far there is no obvious pattern in when the spikes occur.

Next let's investigate the significance of these peak sale days. When do these peaks occur after tickets go on sale? To plot tickets sold per day, select the options shown in the following image in Power BI.



This plot shows that some venues sell large numbers of tickets on the first day of sale. As soon as tickets go on sale at these venues, there seems to be a mad rush. This burst of activity by a few venues might impact the service for other tenants.

You can drill into the data again to see if this mad rush is true for all events hosted by these venues. In previous plots, you saw that Contoso Concert Hall sells many tickets, and that Contoso also has a spike in ticket sales on certain days. Play around with Power BI options to plot cumulative ticket sales for Contoso Concert Hall, focusing on sale trends for each of its events. Do all events follow the same sale pattern? Try to produce a plot like the one below.



This plot of cumulative ticket sales over time for Contoso Concert Hall for each event shows that the mad rush does not happen for all events. Play around with the filter options to explore sale trends for other venues.

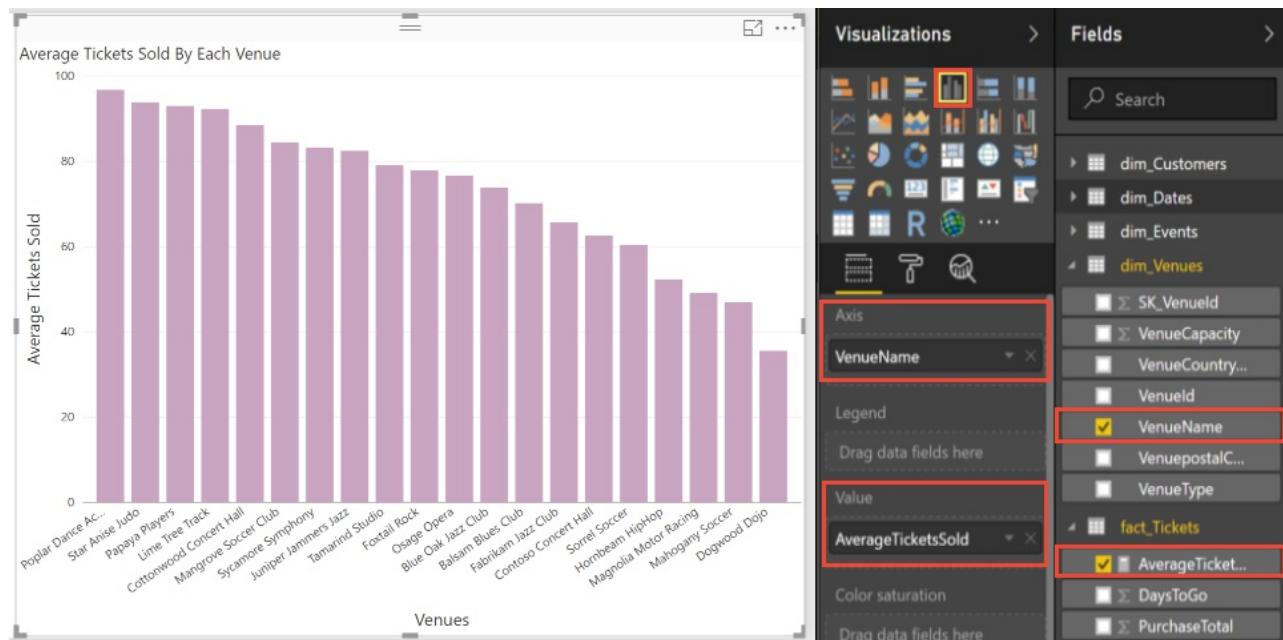
The insights into ticket selling patterns might lead Wingtip Tickets to optimize their business model. Instead of charging all tenants equally, perhaps Wingtip should introduce service tiers with different compute sizes. Larger venues that need to sell more tickets per day could be offered a higher tier with a higher service level agreement.

(SLA). Those venues could have their databases placed in pool with higher per-database resource limits. Each service tier could have an hourly sales allocation, with additional fees charged for exceeding the allocation. Larger venues that have periodic bursts of sales would benefit from the higher tiers, and Wingtip Tickets can monetize their service more efficiently.

Meanwhile, some Wingtip Tickets customers complain that they struggle to sell enough tickets to justify the service cost. Perhaps in these insights there is an opportunity to boost ticket sales for underperforming venues. Higher sales would increase the perceived value of the service. Right click fact\_Tickets and select **New measure**. Enter the following expression for the new measure called **Average Tickets Sold**:

```
AverageTicketsSold = DIVIDE(DIVIDE(COUNTROWS(fact_Tickets),DISTINCT(dim_Venues[VenueCapacity]))*100,
COUNTROWS(dim_Events))
```

Select the following visualization options to plot the percentage tickets sold by each venue to determine their relative success.



The plot above shows that even though most venues sell more than 80% of their tickets, some are struggling to fill more than half their seats. Play around with the Values Well to select maximum or minimum percentage of tickets sold for each venue.

## Embedding analytics in your apps

This tutorial has focused on cross-tenant analytics used to improve the software vendor's understanding of their tenants. Analytics can also provide insights to the *tenants*, to help them manage their business more effectively themselves.

In the Wingtip Tickets example, you earlier discovered that ticket sales tend to follow predictable patterns. This insight might be used to help underperforming venues boost ticket sales. Perhaps there is an opportunity to employ machine learning techniques to predict ticket sales for events. The effects of price changes could also be modeled, to allow the impact of offering discounts to be predicted. Power BI Embedded could be integrated into an event management application to visualize predicted sales, including the impact of discounts on total seats sold and revenue on low-selling events. With Power BI Embedded, you can even integrate actually applying the discount to the ticket prices, right in the visualization experience.

## Next steps

In this tutorial, you learned how to:

- Deploy a SQL Data Warehouse populated with a star schema for tenant analytics.
- Use Azure Data Factory to extract data from each tenant database into the analytics data warehouse.
- Optimize the extracted data (reorganize into a star-schema).
- Query the analytics data warehouse.
- Use Power BI to visualize trends in data across all the tenants.

Congratulations!

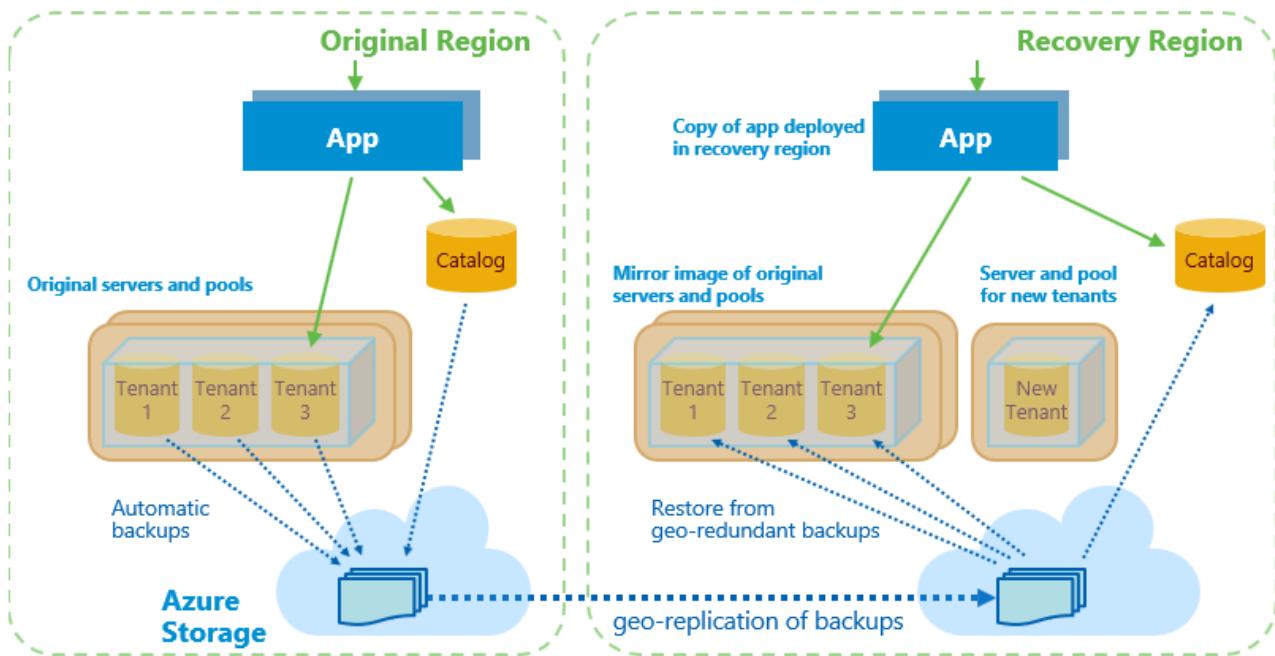
## Additional resources

- Additional [tutorials that build upon the Wingtip SaaS application](#).

# Use geo-restore to recover a multitenant SaaS application from database backups

11/7/2019 • 18 minutes to read • [Edit Online](#)

This tutorial explores a full disaster recovery scenario for a multitenant SaaS application implemented with the database per tenant model. You use [geo-restore](#) to recover the catalog and tenant databases from automatically maintained geo-redundant backups into an alternate recovery region. After the outage is resolved, you use [geo-replication](#) to repatriate changed databases to their original region.



Geo-restore is the lowest-cost disaster recovery solution for Azure SQL Database. However, restoring from geo-redundant backups can result in data loss of up to one hour. It can take considerable time, depending on the size of each database.

## NOTE

Recover applications with the lowest possible RPO and RTO by using geo-replication instead of geo-restore.

This tutorial explores both restore and repatriation workflows. You learn how to:

- Sync database and elastic pool configuration info into the tenant catalog.
- Set up a mirror image environment in a recovery region that includes application, servers, and pools.
- Recover catalog and tenant databases by using geo-restore.
- Use geo-replication to repatriate the tenant catalog and changed tenant databases after the outage is resolved.
- Update the catalog as each database is restored (or repatriated) to track the current location of the active copy of each tenant's database.
- Ensure that the application and tenant database are always co-located in the same Azure region to reduce latency.

Before you start this tutorial, complete the following prerequisites:

- Deploy the Wingtip Tickets SaaS database per tenant app. To deploy in less than five minutes, see [Deploy and](#)

explore the Wingtip Tickets SaaS database per tenant application.

- Install Azure PowerShell. For details, see [Getting started with Azure PowerShell](#).

## Introduction to the geo-restore recovery pattern

Disaster recovery (DR) is an important consideration for many applications, whether for compliance reasons or business continuity. If there's a prolonged service outage, a well-prepared DR plan can minimize business disruption. A DR plan based on geo-restore must accomplish several goals:

- Reserve all needed capacity in the chosen recovery region as quickly as possible to ensure that it's available to restore tenant databases.
- Establish a mirror image recovery environment that reflects the original pool and database configuration.
- Allow cancellation of the restore process in mid-flight if the original region comes back online.
- Enable tenant provisioning quickly so new tenant onboarding can restart as soon as possible.
- Be optimized to restore tenants in priority order.
- Be optimized to get tenants online as soon as possible by doing steps in parallel where practical.
- Be resilient to failure, restartable, and idempotent.
- Repatriate databases to their original region with minimal impact to tenants when the outage is resolved.

### NOTE

The application is recovered into the paired region of the region in which the application is deployed. For more information, see [Azure paired regions](#).

This tutorial uses features of Azure SQL Database and the Azure platform to address these challenges:

- [Azure Resource Manager templates](#), to reserve all needed capacity as quickly as possible. Azure Resource Manager templates are used to provision a mirror image of the original servers and elastic pools in the recovery region. A separate server and pool are also created for provisioning new tenants.
- [Elastic Database Client Library](#) (EDCL), to create and maintain a tenant database catalog. The extended catalog includes periodically refreshed pool and database configuration information.
- [Shard management recovery features](#) of the EDCL, to maintain database location entries in the catalog during recovery and repatriation.
- [Geo-restore](#), to recover the catalog and tenant databases from automatically maintained geo-redundant backups.
- [Asynchronous restore operations](#), sent in tenant-priority order, are queued for each pool by the system and processed in batches so the pool isn't overloaded. These operations can be canceled before or during execution if necessary.
- [Geo-replication](#), to repatriate databases to the original region after the outage. There is no data loss and minimal impact on the tenant when you use geo-replication.
- [SQL server DNS aliases](#), to allow the catalog sync process to connect to the active catalog regardless of its location.

## Get the disaster recovery scripts

The DR scripts used in this tutorial are available in the [Wingtip Tickets SaaS database per tenant GitHub repository](#). Check out the [general guidance](#) for steps to download and unlock the Wingtip Tickets management scripts.

## IMPORTANT

Like all the Wingtip Tickets management scripts, the DR scripts are sample quality and are not to be used in production.

## Review the healthy state of the application

Before you start the recovery process, review the normal healthy state of the application.

1. In your web browser, open the Wingtip Tickets events hub (<http://events.wingtip-dpt.<user>.trafficmanager.net>, replace <user> with your deployment's user value).

Scroll to the bottom of the page and notice the catalog server name and location in the footer. The location is the region in which you deployed the app.

### TIP

Hover the mouse over the location to enlarge the display.

The screenshot shows the Wingtip Tickets Platform Events Hub. At the top, there's a navigation bar with a ticket icon and the text "Wingtip Tickets Platform". Below it, a dark header bar says "Events Hub". The main content area has a title "Welcome to the Wingtip Tickets Platform!". Below the title, a message encourages users to explore venues and purchase tickets. A search bar labeled "search" is followed by a dropdown menu titled "Venues" containing four items: "Contoso Concert Hall", "Dogwood Dojo", and "Fabrikam Jazz Club". At the bottom of this section, a red box highlights the footer text: "Catalog database: tenantcatalog", "Server: catalog-dpt-bgdr1", and "Location: southcentralus". The footer also includes a link to "Learn how to build a SaaS app on SQL database".

2. Select the Contoso Concert Hall tenant and open its event page.

In the footer, notice the tenant's server name. The location is the same as the catalog server's location.

The screenshot shows the event page for "An Evening with Tchaikovsky" at the "Contoso Symphony" venue. The date is listed as "MAR 26 MON". The event title is "An Evening with Tchaikovsky" with a "Tickets" button. Below the title, it says "Contoso Symphony". The footer of the page includes a ticket icon and the text "Contoso Concert Hall". A red box highlights the footer text: "Tenant Id: 1976168774 | raw: 0xF5C9F146", "Database: contosoconcerthall", "Server: tenants1-dpt-bgdr1", and "Location: southcentralus". The footer also includes a link to "Learn how to build a SaaS app on SQL database".

3. In the [Azure portal](#), review and open the resource group in which you deployed the app.

Notice the resources and the region in which the app service components and SQL Database servers are deployed.

## Sync the tenant configuration into the catalog

In this task, you start a process to sync the configuration of the servers, elastic pools, and databases into the tenant catalog. This information is used later to configure a mirror image environment in the recovery region.

### IMPORTANT

For simplicity, the sync process and other long-running recovery and repatriation processes are implemented in these samples as local PowerShell jobs or sessions that run under your client user login. The authentication tokens issued when you log in expire after several hours, and the jobs will then fail. In a production scenario, long-running processes should be implemented as reliable Azure services of some kind, running under a service principal. See [Use Azure PowerShell to create a service principal with a certificate](#).

1. In the PowerShell ISE, open the ...\\Learning Modules\\UserConfig.psm1 file. Replace `<resourcegroup>` and `<user>` on lines 10 and 11 with the value used when you deployed the app. Save the file.
2. In the PowerShell ISE, open the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\Demo-RestoreFromBackup.ps1 script.

In this tutorial, you run each of the scenarios in this PowerShell script, so keep this file open.

3. Set the following:

```
$DemoScenario = 1: Start a background job that syncs tenant server and pool configuration info into the catalog.
```

4. To run the sync script, select F5.

This information is used later to ensure that recovery creates a mirror image of the servers, pools, and databases in the recovery region.

```
Windows PowerShell
```

```
Acquired tenant catalog: 'catalog-dpt-bgdr-recovery/tenantcatalog'
Synchronizing tenant servers...
Synchronizing tenant elastic pools...
Synchronizing tenant databases...
Sleeping for 55 seconds

```

Leave the PowerShell window running in the background and continue with the rest of this tutorial.

### NOTE

The sync process connects to the catalog via a DNS alias. The alias is modified during restore and repatriation to point to the active catalog. The sync process keeps the catalog up to date with any database or pool configuration changes made in the recovery region. During repatriation, these changes are applied to the equivalent resources in the original region.

## Geo-restore recovery process overview

The geo-restore recovery process deploys the application and restores databases from backups into the recovery region.

The recovery process does the following:

1. Disables the Azure Traffic Manager endpoint for the web app in the original region. Disabling the endpoint prevents users from connecting to the app in an invalid state should the original region come online during recovery.
2. Provisions a recovery catalog server in the recovery region, geo-restores the catalog database, and updates the activecatalog alias to point to the restored catalog server. Changing the catalog alias ensures that the catalog sync process always syncs to the active catalog.
3. Marks all existing tenants in the recovery catalog as offline to prevent access to tenant databases before they are restored.
4. Provisions an instance of the app in the recovery region and configures it to use the restored catalog in that region. To keep latency to a minimum, the sample app is designed to always connect to a tenant database in the same region.
5. Provisions a server and elastic pool in which new tenants are provisioned. Creating these resources ensures that provisioning new tenants doesn't interfere with the recovery of existing tenants.
6. Updates the new tenant alias to point to the server for new tenant databases in the recovery region. Changing this alias ensures that databases for any new tenants are provisioned in the recovery region.
7. Provisions servers and elastic pools in the recovery region for restoring tenant databases. These servers and pools are a mirror image of the configuration in the original region. Provisioning pools up front reserves the capacity needed to restore all the databases.

An outage in a region might place significant pressure on the resources available in the paired region. If you rely on geo-restore for DR, then reserving resources quickly is recommended. Consider geo-replication if it's critical that an application is recovered in a specific region.

8. Enables the Traffic Manager endpoint for the web app in the recovery region. Enabling this endpoint allows the application to provision new tenants. At this stage, existing tenants are still offline.
9. Submits batches of requests to restore databases in priority order.
  - Batches are organized so that databases are restored in parallel across all pools.
  - Restore requests are submitted asynchronously so they are submitted quickly and queued for execution in each pool.
  - Because restore requests are processed in parallel across all pools, it's better to distribute important tenants across many pools.
10. Monitors the SQL Database service to determine when databases are restored. After a tenant database is restored, it's marked online in the catalog, and a rowversion sum for the tenant database is recorded.
  - Tenant databases can be accessed by the application as soon as they're marked online in the catalog.
  - A sum of rowversion values in the tenant database is stored in the catalog. This sum acts as a fingerprint that allows the repatriation process to determine if the database was updated in the recovery region.

## Run the recovery script

## IMPORTANT

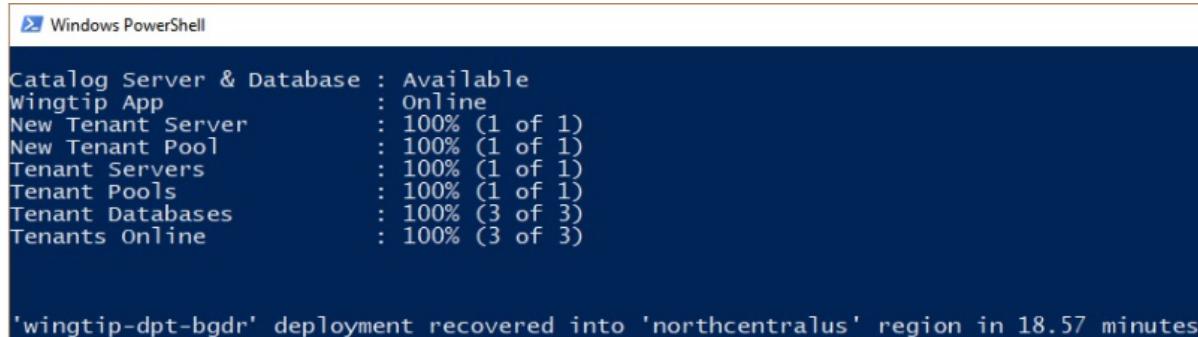
This tutorial restores databases from geo-redundant backups. Although these backups are typically available within 10 minutes, it can take up to an hour. The script pauses until they're available.

Imagine there's an outage in the region in which the application is deployed, and run the recovery script:

1. In the PowerShell ISE, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\Demo-RestoreFromBackup.ps1 script, set the following value:  

```
$DemoScenario = 2: Recover the app into a recovery region by restoring from geo-redundant backups.
```

2. To run the script, select F5.
  - The script opens in a new PowerShell window and then starts a set of PowerShell jobs that run in parallel. These jobs restore servers, pools, and databases to the recovery region.
  - The recovery region is the paired region associated with the Azure region in which you deployed the application. For more information, see [Azure paired regions](#).
3. Monitor the status of the recovery process in the PowerShell window.



```
Windows PowerShell

Catalog Server & Database : Available
Wingtip App : Online
New Tenant Server : 100% (1 of 1)
New Tenant Pool : 100% (1 of 1)
Tenant Servers : 100% (1 of 1)
Tenant Pools : 100% (1 of 1)
Tenant Databases : 100% (3 of 3)
Tenants Online : 100% (3 of 3)

'wingtip-dpt-bgdr' deployment recovered into 'northcentralus' region in 18.57 minutes
```

## NOTE

To explore the code for the recovery jobs, review the PowerShell scripts in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\RecoveryJobs folder.

## Review the application state during recovery

While the application endpoint is disabled in Traffic Manager, the application is unavailable. The catalog is restored, and all the tenants are marked offline. The application endpoint in the recovery region is then enabled, and the application is back online. Although the application is available, tenants appear offline in the events hub until their databases are restored. It's important to design your application to handle offline tenant databases.

- After the catalog database has been recovered but before the tenants are back online, refresh the Wingtip Tickets events hub in your web browser.
  - In the footer, notice that the catalog server name now has a -recovery suffix and is located in the recovery region.
  - Notice that tenants that are not yet restored are marked as offline and are not selectable.



## Events Hub

**Welcome to the Wingtip Tickets Platform!**

Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database

search

**Venues**

Contoso Concert Hall (offline)

Dogwood Dojo (offline)

Fabrikam Jazz Club (offline)

1 | End



Running on Azure SQL Database. © 2018 Microsoft

Catalog database: tenantcatalog  
Server: catalog-dpt-bgdr1-recovery  
Location: northcentralus

Learn how to build a SaaS app on SQL database

- If you open a tenant's events page directly while the tenant is offline, the page displays a tenant offline notification. For example, if Contoso Concert Hall is offline, try to open <http://events.wingtip-dpt.<user>.trafficmanager.net/contosoconcerthall>.

**Contoso Concert Hall****We'll be back soon!**

Sorry for the inconvenience but we're performing some maintenance at the moment. We'll be back online shortly but if you need to, you can contact us via email.

— Contoso Concert Hall

**Provision a new tenant in the recovery region**

Even before tenant databases are restored, you can provision new tenants in the recovery region. New tenant databases provisioned in the recovery region are repatriated with the recovered databases later.

1. In the PowerShell ISE, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\Demo-RestoreFromBackup.ps1 script, set the following property:

```
$DemoScenario = 3: Provision a new tenant in the recovery region.
```

2. To run the script, select F5.
3. The Hawthorn Hall events page opens in the browser when provisioning finishes.

Notice that the Hawthorn Hall database is located in the recovery region.



Next | Event 3 Tickets >

<p>MAR 20 TUE</p> <p>Event 3 Performer 3</p>	<p>MAR 23 FRI</p> <p>Event 4 Performer 4</p>	<p>Tickets</p>	<p><b>mytickets</b> Update your list of favorites and never miss an event! <a href="#">Sign In</a></p>
----------------------------------------------------------	----------------------------------------------------------	----------------	----------------------------------------------------------------------------------------------------------------

**Hawthorn Hall**

Running on Azure SQL Database. © 2018 Microsoft

Tenant id: 348342172 | raw: 0x94C3479C  
Database: hawthornhall  
Server: tenants2-dpt-bydr1-recovery  
Location: northcentralus

Learn how to build a SaaS app on SQL database

4. In the browser, refresh the Wingtip Tickets events hub page to see Hawthorn Hall included.

If you provisioned Hawthorn Hall without waiting for the other tenants to restore, other tenants might still be offline.

## Review the recovered state of the application

When the recovery process finishes, the application and all tenants are fully functional in the recovery region.

1. After the display in the PowerShell console window indicates all the tenants are recovered, refresh the events hub.

The tenants all appear online, including the new tenant, Hawthorn Hall.



## Events Hub

## Welcome to the Wingtip Tickets Platform!

Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database

**Venues**

Contoso Concert Hall

Dogwood Dojo

Fabrikam Jazz Club

Hawthorn Hall

1 | End

- Click on Contoso Concert Hall and open its events page.

In the footer, notice that the database is located on the recovery server located in the recovery region.

MAR  
26  
MON

An Evening with Tchaikovsky Tickets

Contoso Symphony

Contoso Concert Hall

Running on Azure SQL Database. © 2018 Microsoft

Tenant id: 1976168774 | raw: 0xF5C9F146  
Database: contosoconcerthall  
Server: tenants1-dpt-bgdr1-recovery  
Location: northcentralus

Learn how to build a SaaS app on SQL database

- In the [Azure portal](#), open the list of resource groups.

Notice the resource group that you deployed, plus the recovery resource group, with the -recovery suffix. The recovery resource group contains all the resources created during the recovery process, plus new resources created during the outage.

- Open the recovery resource group and notice the following items:

- The recovery versions of the catalog and tenants1 servers, with the -recovery suffix. The restored catalog and tenant databases on these servers all have the names used in the original region.
- The tenants2-dpt-<user>-recovery SQL server. This server is used for provisioning new tenants during the outage.
- The app service named events-wingtip-dpt-<recoveryregion>-<user>, which is the recovery instance of the events app.

NAME	TYPE	LOCATION
catalog-dpt-bgdr-recovery	SQL server	North Central US
baseTenantDB	SQL database	North Central US
tenantcatalog	SQL database	North Central US
events-wingtip-dpt-northcentralus-bgdr	App Service plan	North Central US
events-wingtip-dpt-northcentralus-bgdr	App Service	North Central US
tenants1-dpt-bgdr-recovery	SQL server	North Central US
contosoconcerthall	SQL database	North Central US
dogwooddojo	SQL database	North Central US
fabrikamjazzclub	SQL database	North Central US
Pool1	SQL elastic pool	North Central US
tenants2-dpt-bgdr-recovery	SQL server	North Central US
hawthornhall	SQL database	North Central US
Pool1	SQL elastic pool	North Central US

5. Open the tenants2-dpt-<user>-recovery SQL server. Notice that it contains the database hawthornhall and the elastic pool Pool1. The hawthornhall database is configured as an elastic database in the Pool1 elastic pool.

## Change the tenant data

In this task, you update one of the restored tenant databases. The repatriation process copies restored databases that have been changed to the original region.

1. In your browser, find the events list for the Contoso Concert Hall, scroll through the events, and notice the last event, Seriously Strauss.
2. In the PowerShell ISE, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\Demo-RestoreFromBackup.ps1 script, set the following value:  

```
$DemoScenario = 4: Delete an event from a tenant in the recovery region.
```
3. To execute the script, select F5.
4. Refresh the Contoso Concert Hall events page (<http://events.wingtip-dpt-<user>.trafficmanager.net/contosoconcerthall>), and notice that the event Seriously Strauss is missing.

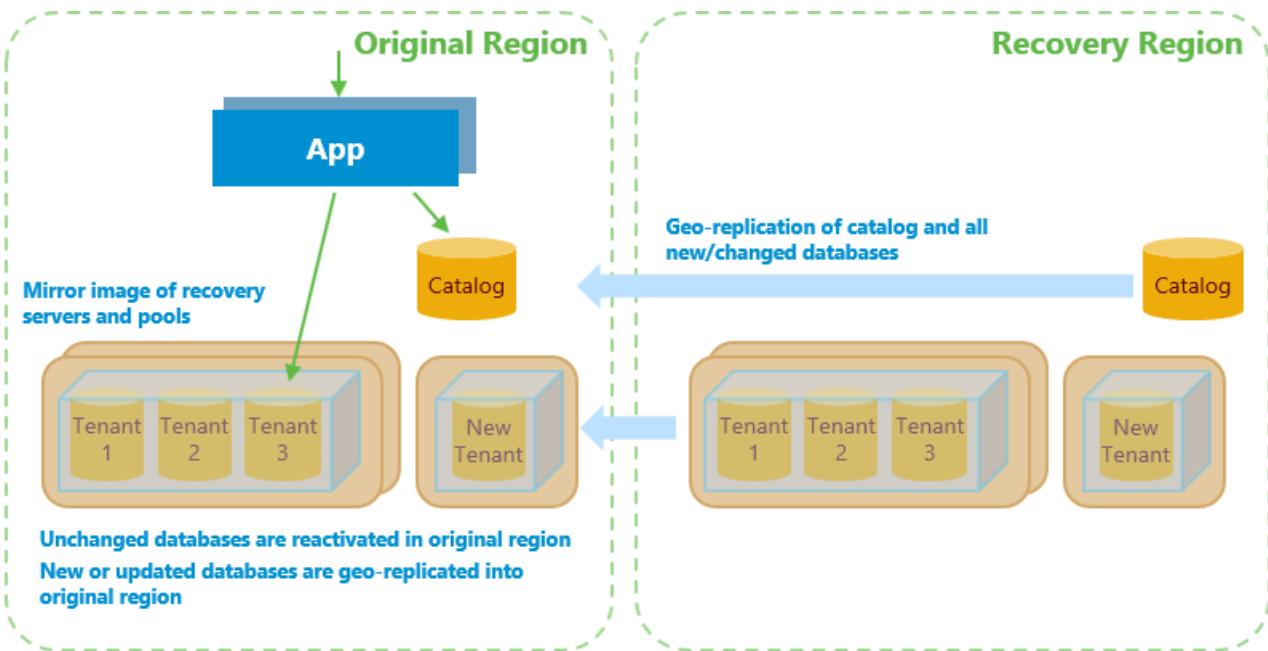
At this point in the tutorial, you have recovered the application, which is now running in the recovery region. You have provisioned a new tenant in the recovery region and modified data of one of the restored tenants.

### NOTE

Other tutorials in the sample are not designed to run with the app in the recovery state. If you want to explore other tutorials, be sure to repatriate the application first.

## Repatriation process overview

The repatriation process reverts the application and its databases to its original region after an outage is resolved.



The process:

1. Stops any ongoing restore activity and cancels any outstanding or in-flight database restore requests.
2. Reactivates in the original region tenant databases that have not been changed since the outage. These databases include those not recovered yet and those recovered but not changed afterward. The reactivated databases are exactly as last accessed by their tenants.
3. Provisions a mirror image of the new tenant's server and elastic pool in the original region. After this action is complete, the new tenant alias is updated to point to this server. Updating the alias causes new tenant onboarding to occur in the original region instead of the recovery region.
4. Uses geo-replication to move the catalog to the original region from the recovery region.
5. Updates pool configuration in the original region so it's consistent with changes that were made in the recovery region during the outage.
6. Creates the required servers and pools to host any new databases created during the outage.
7. Uses geo-replication to repatriate restored tenant databases that have been updated post-restore and all new tenant databases provisioned during the outage.
8. Cleans up resources created in the recovery region during the restore process.

To limit the number of tenant databases that need to be repatriated, steps 1 to 3 are done promptly.

Step 4 is only done if the catalog in the recovery region has been modified during the outage. The catalog is updated if new tenants are created or if any database or pool configuration is changed in the recovery region.

It's important that step 7 causes minimal disruption to tenants and no data is lost. To achieve this goal, the process uses geo-replication.

Before each database is geo-replicated, the corresponding database in the original region is deleted. The database in the recovery region is then geo-replicated, creating a secondary replica in the original region. After replication is complete, the tenant is marked offline in the catalog, which breaks any connections to the database in the recovery region. The database is then failed over, causing any pending transactions to process on the secondary so no data is lost.

On failover, the database roles are reversed. The secondary in the original region becomes the primary read-write database, and the database in the recovery region becomes a read-only secondary. The tenant entry in the catalog is updated to reference the database in the original region, and the tenant is marked online. At this point,

repatriation of the database is complete.

Applications should be written with retry logic to ensure that they reconnect automatically when connections are broken. When they use the catalog to broker the reconnection, they connect to the repatriated database in the original region. Although the brief disconnect is often not noticed, you might choose to repatriate databases out of business hours.

After a database is repatriated, the secondary database in the recovery region can be deleted. The database in the original region then relies again on geo-restore for DR protection.

In step 8, resources in the recovery region, including the recovery servers and pools, are deleted.

## Run the repatriation script

Let's imagine the outage is resolved and run the repatriation script.

If you've followed the tutorial, the script immediately reactivates Fabrikam Jazz Club and Dogwood Dojo in the original region because they're unchanged. It then repatriates the new tenant, Hawthorn Hall, and Contoso Concert Hall because it has been modified. The script also repatriates the catalog, which was updated when Hawthorn Hall was provisioned.

1. In the PowerShell ISE, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\Demo-RestoreFromBackup.ps1 script, verify that the Catalog Sync process is still running in its PowerShell instance. If necessary, restart it by setting:

```
$DemoScenario = 1: Start synchronizing tenant server, pool, and database configuration info into the catalog.
```

To run the script, select F5.

2. Then to start the repatriation process, set:

```
$DemoScenario = 5: Repatriate the app into its original region.
```

To run the recovery script in a new PowerShell window, select F5. Repatriation takes several minutes and can be monitored in the PowerShell window.

3. While the script is running, refresh the events hub page (<http://events.wingtip-dpt.<user>.trafficmanager.net>).

Notice that all the tenants are online and accessible throughout this process.

4. Select the Fabrikam Jazz Club to open it. If you didn't modify this tenant, notice from the footer that the server is already reverted to the original server.
5. Open or refresh the Contoso Concert Hall events page. Notice from the footer that, initially, the database is still on the -recovery server.
6. Refresh the Contoso Concert Hall events page when the repatriation process finishes, and notice that the database is now in your original region.
7. Refresh the events hub again and open Hawthorn Hall. Notice that its database is also located in the original region.

## Clean up recovery region resources after repatriation

After repatriation is complete, it's safe to delete the resources in the recovery region.

**IMPORTANT**

Delete these resources promptly to stop all billing for them.

The restore process creates all the recovery resources in a recovery resource group. The cleanup process deletes this resource group and removes all references to the resources from the catalog.

1. In the PowerShell ISE, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-RestoreFromBackup\\Demo-RestoreFromBackup.ps1 script, set:

```
$DemoScenario = 6: Delete obsolete resources from the recovery region.
```

2. To run the script, select F5.

After cleaning up the scripts, the application is back where it started. At this point, you can run the script again or try out other tutorials.

## Designing the application to ensure that the app and the database are co-located

The application is designed to always connect from an instance in the same region as the tenant's database. This design reduces latency between the application and the database. This optimization assumes the app-to-database interaction is chattier than the user-to-app interaction.

Tenant databases might be spread across recovery and original regions for some time during repatriation. For each database, the app looks up the region in which the database is located by doing a DNS lookup on the tenant server name. In SQL Database, the server name is an alias. The aliased server name contains the region name. If the application isn't in the same region as the database, it redirects to the instance in the same region as the database server. Redirecting to the instance in the same region as the database minimizes latency between the app and the database.

## Next steps

In this tutorial, you learned how to:

- Use the tenant catalog to hold periodically refreshed configuration information, which allows a mirror image recovery environment to be created in another region.
- Recover Azure SQL databases into the recovery region by using geo-restore.
- Update the tenant catalog to reflect restored tenant database locations.
- Use a DNS alias to enable an application to connect to the tenant catalog throughout without reconfiguration.
- Use geo-replication to repatriate recovered databases to their original region after an outage is resolved.

Try the [Disaster recovery for a multitenant SaaS application using database geo-replication](#) tutorial to learn how to use geo-replication to dramatically reduce the time needed to recover a large-scale multitenant application.

## Additional resources

[Additional tutorials that build upon the Wingtip SaaS application](#)

# Disaster recovery for a multi-tenant SaaS application using database geo-replication

11/7/2019 • 17 minutes to read • [Edit Online](#)

In this tutorial, you explore a full disaster recovery scenario for a multi-tenant SaaS application implemented using the database-per-tenant model. To protect the app from an outage, you use *geo-replication* to create replicas for the catalog and tenant databases in an alternate recovery region. If an outage occurs, you quickly fail over to these replicas to resume normal business operations. On failover, the databases in the original region become secondary replicas of the databases in the recovery region. Once these replicas come back online they automatically catch up to the state of the databases in the recovery region. After the outage is resolved, you fail back to the databases in the original production region.

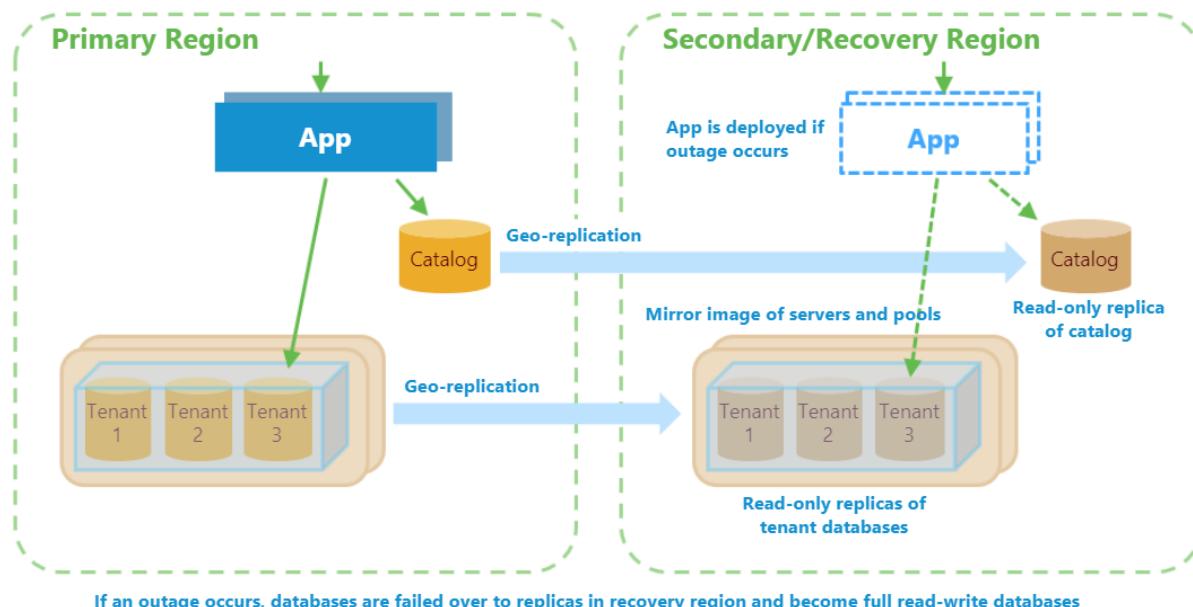
This tutorial explores both the failover and fallback workflows. You'll learn how to:

- Sync database and elastic pool configuration info into the tenant catalog
- Set up a recovery environment in an alternate region, comprising application, servers, and pools
- Use *geo-replication* to replicate the catalog and tenant databases to the recovery region
- Fail over the application and catalog and tenant databases to the recovery region
- Later, fail over the application, catalog and tenant databases back to the original region after the outage is resolved
- Update the catalog as each tenant database is failed over to track the primary location of each tenant's database
- Ensure the application and primary tenant database are always colocated in the same Azure region to reduce latency

Before starting this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS database per tenant app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS database per tenant application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)

## Introduction to the geo-replication recovery pattern



Disaster recovery (DR) is an important consideration for many applications, whether for compliance reasons or business continuity. Should there be a prolonged service outage, a well-prepared DR plan can minimize business disruption. Using geo-replication provides the lowest RPO and RTO by maintaining database replicas in a recovery region that can be failed over to at short notice.

A DR plan based on geo-replication comprises three distinct parts:

- Set-up - creation and maintenance of the recovery environment
- Recovery - failover of the app and databases to the recovery environment if an outage occurs,
- Repatriation - failover of the app and databases back to the original region once the application is resolved

All parts have to be considered carefully, especially if operating at scale. Overall, the plan must accomplish several goals:

- Setup
  - Establish and maintain a mirror-image environment in the recovery region. Creating the elastic pools and replicating any databases in this recovery environment reserves capacity in the recovery region. Maintaining this environment includes replicating new tenant databases as they are provisioned.
- Recovery
  - Where a scaled-down recovery environment is used to minimize day-to-day costs, pools and databases must be scaled up to acquire full operational capacity in the recovery region
  - Enable new tenant provisioning in the recovery region as soon as possible
  - Be optimized for restoring tenants in priority order
  - Be optimized for getting tenants online as fast as possible by doing steps in parallel where practical
  - Be resilient to failure, restartable, and idempotent
  - Be possible to cancel the process in mid-flight if the original region comes back on-line.
- Repatriation
  - Fail over databases from the recovery region to replicas in the original region with minimal impact to tenants: no data loss and minimum period off-line per tenant.

In this tutorial, these challenges are addressed using features of Azure SQL Database and the Azure platform:

- [Azure Resource Manager templates](#), to reserve all needed capacity as quickly as possible. Azure Resource Manager templates are used to provision a mirror image of the production servers and elastic pools in the recovery region.
- [Geo-replication](#), to create asynchronously replicated read-only secondaries for all databases. During an outage, you fail over to the replicas in the recovery region. After the outage is resolved, you fail back to the databases in the original region with no data loss.
- [Asynchronous failover operations](#) sent in tenant-priority order, to minimize failover time for large numbers of databases.
- [Shard management recovery features](#), to change database entries in the catalog during recovery and repatriation. These features allow the app to connect to tenant databases regardless of location without reconfiguring the app.
- [SQL server DNS aliases](#), to enable seamless provisioning of new tenants regardless of which region the app is operating in. DNS aliases are also used to allow the catalog sync process to connect to the active catalog regardless of its location.

## Get the disaster recovery scripts

### IMPORTANT

Like all the Wingtip Tickets management scripts, the DR scripts are sample quality and are not to be used in production.

The recovery scripts used in this tutorial and Wingtip application source code are available in the [Wingtip Tickets SaaS database per tenant GitHub repository](#). Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets management scripts.

## Tutorial overview

In this tutorial, you first use geo-replication to create replicas of the Wingtip Tickets application and its databases in a different region. Then, you fail over to this region to simulate recovering from an outage. When complete, the application is fully functional in the recovery region.

Later, in a separate repatriation step, you fail over the catalog and tenant databases in the recovery region to the original region. The application and databases stay available throughout repatriation. When complete, the application is fully functional in the original region.

### NOTE

The application is recovered into the *paired region* of the region in which the application is deployed. For more information, see [Azure paired regions](#).

## Review the healthy state of the application

Before you start the recovery process, review the normal healthy state of the application.

1. In your web browser, open the Wingtip Tickets Events Hub (<http://events.wingtip-dpt.<user>.trafficmanager.net> - replace <user> with your deployment's user value).
  - Scroll to the bottom of the page and notice the catalog server name and location in the footer. The location is the region in which you deployed the app. *TIP: Hover the mouse over the location to enlarge the display.*

The screenshot shows the Wingtip Tickets Platform Events Hub. At the top, there's a header with a ticket icon and the text "Wingtip Tickets Platform". Below that is a dark grey navigation bar with the text "Events Hub". The main content area has a title "Welcome to the Wingtip Tickets Platform!". Below it, a message says "Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database". There's a search bar labeled "search" and a dropdown menu titled "Venues" containing "Contoso Concert Hall", "Dogwood Dojo", and "Fabrikam Jazz Club". At the bottom of the page, there's a footer with the "Wingtip Tickets Platform" logo, a note that it's "Running on Azure SQL Database. © 2018 Microsoft", and a red-bordered box containing "Catalog database: tenantcatalog", "Server: catalog-dpt-bgdr2", and "Location: northcentralus".

2. Click on the Contoso Concert Hall tenant and open its event page.

- In the footer, notice the tenant server name. The location will be the same as the catalog server's location.
3. In the [Azure portal](#), open the resource group in which the app is deployed
- Notice the region in which the servers are deployed.

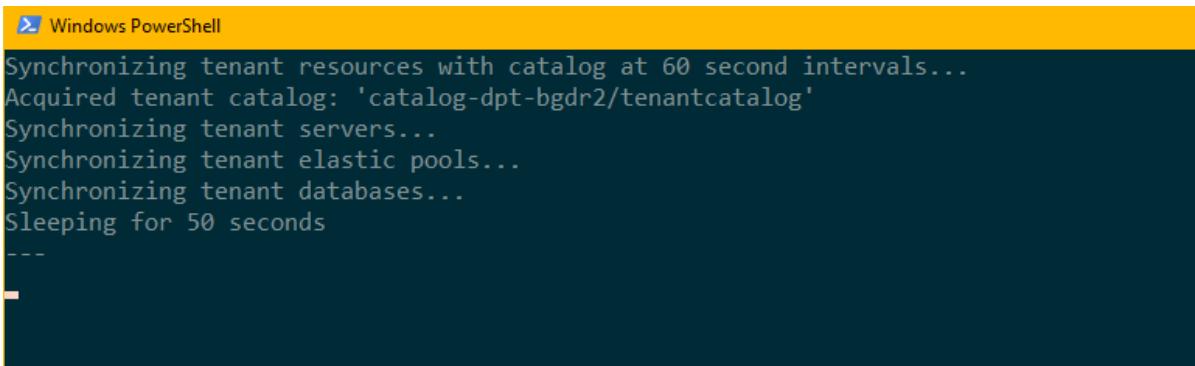
## Sync tenant configuration into catalog

In this task, you start a process that syncs the configuration of the servers, elastic pools, and databases into the tenant catalog. The process keeps this information up-to-date in the catalog. The process works with the active catalog, whether in the original region or in the recovery region. The configuration information is used as part of the recovery process to ensure the recovery environment is consistent with the original environment, and then later during repatriation to ensure the original region is made consistent with any changes made in the recovery environment. The catalog is also used to keep track of the recovery state of tenant resources

### IMPORTANT

For simplicity, the sync process and other long running recovery and repatriation processes are implemented in these tutorials as local PowerShell jobs or sessions that run under your client user login. The authentication tokens issued when you login will expire after several hours and the jobs will then fail. In a production scenario, long-running processes should be implemented as reliable Azure services of some kind, running under a service principal. See [Use Azure PowerShell to create a service principal with a certificate](#).

1. In the *PowerShell ISE*, open the ...\\Learning Modules\\UserConfig.psm1 file. Replace `<resourcegroup>` and `<user>` on lines 10 and 11 with the value used when you deployed the app. Save the file!
2. In the *PowerShell ISE*, open the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\Demo-FailoverToReplica.ps1 script and set:
  - **\$DemoScenario = 1**, Start a background job that syncs tenant server, and pool configuration info into the catalog
3. Press **F5** to run the sync script. A new PowerShell session is opened to sync the configuration of tenant resources.



```
Windows PowerShell
Synchronizing tenant resources with catalog at 60 second intervals...
Acquired tenant catalog: 'catalog-dpt-bgdr2/tenantcatalog'
Synchronizing tenant servers...
Synchronizing tenant elastic pools...
Synchronizing tenant databases...
Sleeping for 50 seconds

```

Leave the PowerShell window running in the background and continue with the rest of the tutorial.

### NOTE

The sync process connects to the catalog via a DNS alias. This alias is modified during restore and repatriation to point to the active catalog. The sync process keeps the catalog up-to-date with any database or pool configuration changes made in the recovery region. During repatriation, these changes are applied to the equivalent resources in the original region.

## Create secondary database replicas in the recovery region

In this task, you start a process that deploys a duplicate app instance and replicates the catalog and all tenant databases to a recovery region.

## **NOTE**

This tutorial adds geo-replication protection to the Wingtip Tickets sample application. In a production scenario for an application that uses geo-replication, each tenant would be provisioned with a geo-replicated database from the outset. See [Designing highly available services using Azure SQL Database](#)

1. In the *PowerShell ISE*, open the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\Demo-FailoverToReplica.ps1 script and set the following values:

- **\$DemoScenario = 2**, Create mirror image recovery environment and replicate catalog and tenant databases

2. Press **F5** to run the script. A new PowerShell session is opened to create the replicas.

```
Windows PowerShell
Refreshing status in 10 seconds...

Wingtip App : Deployed
Management & Tenant Servers : Deployed (2 of 2)
Tenant Pools : Deployed (1 of 1)
Catalog Database(s) : Replicating ... (0 of 2 complete)
Tenant Databases : Replicated (3 of 3)
```

## Review the normal application state

At this point, the application is running normally in the original region and now is protected by geo-replication. Read-only secondary replicas exist in the recovery region for all databases.

1. In the Azure portal, look at your resource groups and note that a resource group has been created with - recovery suffix in the recovery region.
  2. Explore the resources in the recovery resource group.
  3. Click on the Contoso Concert Hall database on the *tenants1-dpt-<user>-recovery* server. Click on Geo-Replication on the left side.

In the Azure regions map, note the geo-replication link between the primary in the original region and the secondary in the recovery region.

## Fail over the application into the recovery region

### Geo-replication recovery process overview

The recovery script performs the following tasks:

1. Disables the Traffic Manager endpoint for the web app in the original region. Disabling the endpoint prevents users from connecting to the app in an invalid state should the original region come online during recovery.
2. Uses a force failover of the catalog database in the recovery region to make it the primary database, and updates the *activecatalog* alias to point to the recovery catalog server.
3. Updates the *newtenant* alias to point to the tenant server in the recovery region. Changing this alias ensures that the databases for any new tenants are provisioned in the recovery region.
4. Marks all existing tenants in the recovery catalog as offline to prevent access to tenant databases before they are failed over.
5. Updates the configuration of all elastic pools and replicated single databases in the recovery region to mirror their configuration in the original region. (This task is only needed if pools or replicated databases in the recovery environment are scaled down during normal operations to reduce costs).
6. Enables the Traffic Manager endpoint for the web app in the recovery region. Enabling this endpoint allows the application to provision new tenants. At this stage, existing tenants are still offline.
7. Submits batches of requests to force fail over databases in priority order.
  - Batches are organized so that databases are failed over in parallel across all pools.

- Failover requests are submitted using asynchronous operations so they are submitted quickly and multiple requests can be processed concurrently.

**NOTE**

In an outage scenario, the primary databases in the original region are offline. Force fail over on the secondary breaks the connection to the primary without trying to apply any residual queued transactions. In a DR drill scenario like this tutorial, if there is any update activity at the time of failover there could be some data loss. Later, during repatriation, when you fail over databases in the recovery region back to the original region, a normal failover is used to ensure there is no data loss.

8. Monitors the SQL database service to determine when databases have been failed over. Once a tenant database is failed over, it updates the catalog to record the recovery state of the tenant database and mark the tenant as online.

- Tenant databases can be accessed by the application as soon as they're marked online in the catalog.
- A sum of rowversion values in the tenant database is stored in the catalog. This value acts as a fingerprint that allows the repatriation process to determine if the database has been updated in the recovery region.

### Run the script to fail over to the recovery region

Now imagine there is an outage in the region in which the application is deployed and run the recovery script:

1. In the *PowerShell ISE*, open the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\Demo-FailoverToReplica.ps1 script and set the following values:
  - **\$DemoScenario = 3**, Recover the app into a recovery region by failing over to replicas
2. Press **F5** to run the script.
  - The script opens in a new PowerShell window and then starts a series of PowerShell jobs that run in parallel. These jobs fail over tenant databases to the recovery region.
  - The recovery region is the *paired region* associated with the Azure region in which you deployed the application. For more information, see [Azure paired regions](#).
3. Monitor the status of the recovery process in the PowerShell window.

```
Windows PowerShell
Refreshing status in 10 seconds...

Tenant databases failed-over into recovery region : 100% (3 of 3)
Tenants online in recovery region : 100% (3 of 3)

'Wingtip-bgdr2' deployment recovered into 'southcentralus' region in 2.85 minutes
```

**NOTE**

To explore the code for the recovery jobs, review the PowerShell scripts in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\RecoveryJobs folder.

### Review the application state during recovery

While the application endpoint is disabled in Traffic Manager, the application is unavailable. After the catalog is failed over to the recovery region and all the tenants marked offline, the application is brought back online.

Although the application is available, each tenant appears offline in the events hub until its database is failed over. It's important to design your application to handle offline tenant databases.

1. Promptly after the catalog database has been recovered, refresh the Wingtip Tickets Events Hub in your web browser.

- In the footer, notice that the catalog server name now has a *-recovery* suffix and is located in the recovery region.
- Notice that tenants that are not yet restored, are marked as offline, and are not selectable.

**NOTE**

With only a few databases to recover, you may not be able to refresh the browser before recovery has completed, so you may not see the tenants while they are offline.

The screenshot shows the Wingtip Tickets Platform Events Hub. At the top, there's a header with a ticket icon and the text "Wingtip Tickets Platform". Below that is a dark grey navigation bar with the text "Events Hub". The main content area has a light grey background and features a heading "Welcome to the Wingtip Tickets Platform!". Below it, a message says "Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database". A search bar contains the word "search". Under the search bar is a section titled "Venues" with a dropdown arrow icon. It lists three items: "Contoso Concert Hall (offline)" with a red box around "(offline)", "Dogwood Dojo (offline)", and "Fabrikam Jazz Club (offline)". At the bottom right of this section, there's a link "1 | End". At the very bottom of the page, there's a footer with a ticket icon and the text "Wingtip Tickets Platform". It also includes the text "Running on Azure SQL Database. © 2018 Microsoft" and a call-to-action "Learn how to build a SaaS app on SQL database". To the right of the footer, there's a red-bordered box containing the text "Catalog database: tenantcatalog", "Server: catalog-dpt-bgdr2-recovery", and "Location: southcentralus".

- If you open an offline tenant's Events page directly, it displays a 'tenant offline' notification. For example, if Contoso Concert Hall is offline, try to open <http://events.wingtip-dpt-<user>.trafficmanager.net/contosoconcerthall>

The screenshot shows the Contoso Concert Hall Events page. At the top, there's a header with a ticket icon and the text "Contoso Concert Hall". The main content area features a large, bold heading "We'll be back soon!". Below it, a message says "Sorry for the inconvenience but we're performing some maintenance at the moment. We'll be back online shortly but if you need to, you can contact us via email." followed by a signature line "— Contoso Concert Hall".

**Provision a new tenant in the recovery region**

Even before all the existing tenant databases have failed over, you can provision new tenants in the recovery region.

1. In the *PowerShell ISE*, open the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\Demo-FailoverToReplica.ps1 script and set the following property:

- **\$DemoScenario = 4**, Provision a new tenant in the recovery region

2. Press **F5** to run the script and provision the new tenant.
3. The Hawthorn Hall events page opens in the browser when it completes. Note from the footer that the Hawthorn Hall database is provisioned in the recovery region.

The screenshot shows the 'Events' section of the Wingtip Tickets application. At the top, there's a navigation bar with 'Next | Event 3' on the left and 'Tickets >' on the right. Below this, three events are listed in a grid:

Date	Event	Performer	Tickets
MAR 31 SAT	Event 3	Performer 3	<a href="#">Tickets</a>
APR 4 WED	Event 4	Performer 4	<a href="#">Tickets</a>
APR 8 SUN	Event 5	Performer 5	<a href="#">Tickets</a>

To the right of the events, there's a sidebar with a 'my|tickets' section containing a 'Sign In' button. At the bottom of the page, there's a footer with a 'Hawthorn Hall' logo, a note about running on Azure SQL Database, and a red-bordered box containing tenant information:

Tenant id: 348342172 | raw: 0x94C3479C  
Database: hawthornhall  
Server: tenants1-dpt-bgdr2-recovery  
Location: southcentralus

Learn how to build a SaaS app on SQL database

4. In the browser, refresh the Wingtip Tickets Events Hub page to see Hawthorn Hall included.
  - If you provisioned Hawthorn Hall without waiting for the other tenants to restore, other tenants may still be offline.

## Review the recovered state of the application

When the recovery process completes, the application and all tenants are fully functional in the recovery region.

1. Once the display in the PowerShell console window indicates all the tenants are recovered, refresh the Events Hub. The tenants will all appear online, including the new tenant, Hawthorn Hall.



## Events Hub

## Welcome to the Wingtip Tickets Platform!

Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database

search

**Venues**

- Contoso Concert Hall
- Dogwood Dojo
- Fabrikam Jazz Club
- Hawthorn Hall

1 | End

2. In the [Azure portal](#), open the list of resource groups.

- Notice the resource group that you deployed, plus the recovery resource group, with the *-recovery* suffix. The recovery resource group contains all the resources created during the recovery process, plus new resources created during the outage.

3. Open the recovery resource group and notice the following items:

- The recovery versions of the catalog and tenants1 servers, with *-recovery* suffix. The restored catalog and tenant databases on these servers all have the names used in the original region.
- The *tenants2-dpt-<user>-recovery* SQL server. This server is used for provisioning new tenants during the outage.
- The App Service named, *events-wingtip-dpt-<recoveryregion>-<user>*, which is the recovery instance of the Events app.

The screenshot shows the Azure Resource Group 'Wingtip-bgdr2-recovery'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Quickstart, Resource costs, Deployments, Policies, Properties, Locks, Automation script, Metrics, Alert rules, and Diagnostics logs. The main pane displays a table of resources under the 'Essentials' tab. The table has columns for NAME, TYPE, and LOCATION. The resources listed are:

NAME	TYPE	LOCATION
catalog-dpt-bgdr2-recovery	SQL server	South Central US
basetenantdb (catalog-dpt-bgdr2-recovery/basetenantdb)	SQL database	South Central US
master (catalog-dpt-bgdr2-recovery/master)	SQL database	South Central US
tenantcatalog (catalog-dpt-bgdr2-recovery/tenantcatalog)	SQL database	South Central US
events-wingtip-dpt-southcentralus-bgdr2	App Service plan	South Central US
events-wingtip-dpt-southcentralus-bgdr2	App Service	South Central US
tenants1-dpt-bgdr2-recovery	SQL server	South Central US
contosoconcerthall (tenants1-dpt-bgdr2-recovery/contosoconcerthall)	SQL database	South Central US
dogwooddojo (tenants1-dpt-bgdr2-recovery/dogwooddojo)	SQL database	South Central US
fabrikamjazzclub (tenants1-dpt-bgdr2-recovery/fabrikamjazzclub)	SQL database	South Central US
hawthornhall (tenants1-dpt-bgdr2-recovery/hawthornhall)	SQL database	South Central US
master (tenants1-dpt-bgdr2-recovery/master)	SQL database	South Central US
Pooll (tenants1-dpt-bgdr2-recovery/Pooll)	SQL elastic pool	South Central US

4. Open the *tenants2-dpt-<user>-recovery* SQL server. Notice it contains the database *hawthornhall* and the

elastic pool, *Pool1*. The *hawthornhall* database is configured as an elastic database in *Pool1* elastic pool.

5. Navigate back to the resource group and click on the Contoso Concert Hall database on the *tenants1-dpt-<user>-recovery* server. Click on Geo-Replication on the left side.

The screenshot shows the Azure portal interface for managing a SQL database named "contosoconcerthall". The left sidebar contains navigation links like Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), and several settings options. The "Geo-Replication" option is selected and highlighted in blue. The main content area displays a world map with numerous green circular icons representing different geographical regions. Below the map, under the heading "PRIMARY", it shows "North Central US" with the identifier "tenants1-dpt-bgdr-recovery/contosoconcerthall". Under the heading "SECONDARIES", it shows "South Central US" with the identifier "tenants1-dpt-bgdr/contosoconcerthall". A note at the top right says, "Select a region on the map or from the Target Regions list to create a secondary database."

## Change tenant data

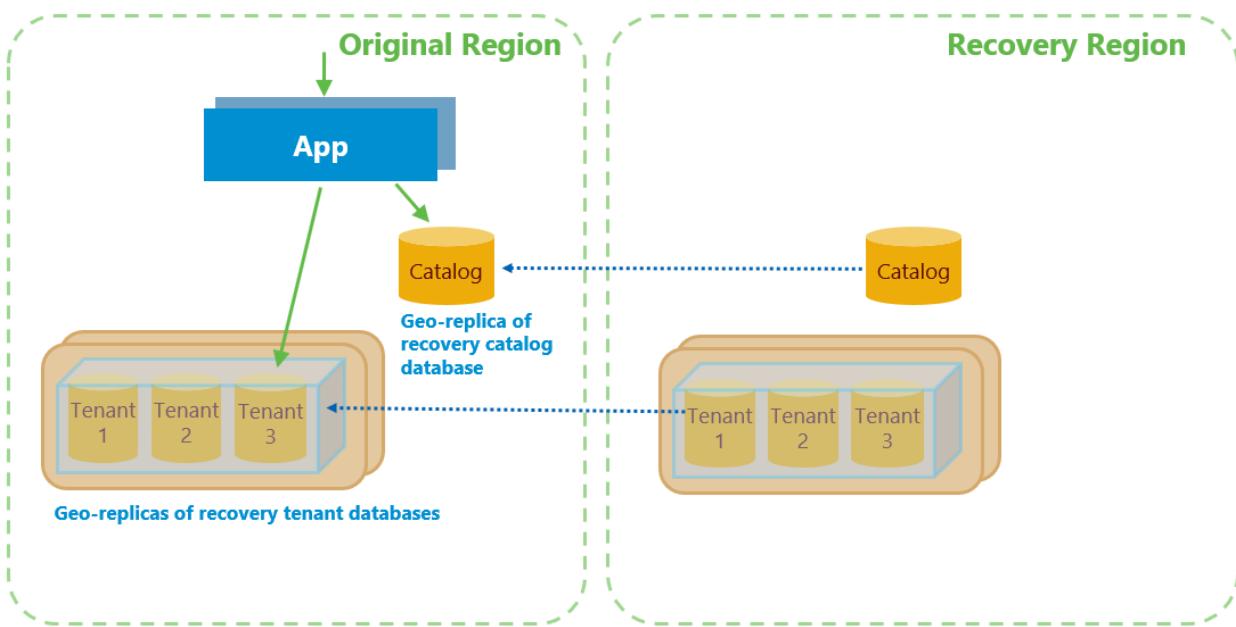
In this task, you update one of the tenant databases.

1. In your browser, find the events list for the Contoso Concert Hall and note the last event name.
2. In the *PowerShell ISE*, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\Demo-FailoverToReplica.ps1 script, set the following value:
  - **\$DemoScenario = 5** Delete an event from a tenant in the recovery region
3. Press **F5** to execute the script
4. Refresh the Contoso Concert Hall events page (<http://events.wingtip-dpt-<user>.trafficmanager.net/contosoconcerthall> - substitute <user> with your deployment's user value) and notice that the last event has been deleted.

## Repatriate the application to its original production region

This task repatriates the application to its original region. In a real scenario, you would initiate repatriation when the outage is resolved.

### Repatriation process overview



The repatriation process:

1. Cancels any outstanding or in-flight database restore requests.
2. Updates the *newtenant* alias to point to the tenants' server in the origin region. Changing this alias ensures that the databases for any new tenants will now be provisioned in the origin region.
3. Seeds any changed tenant data to the original region
4. Fails over tenant databases in priority order.

Failover effectively moves the database to the original region. When the database fails over, any open connections are dropped and the database is unavailable for a few seconds. Applications should be written with retry logic to ensure they connect again. Although this brief disconnect is often not noticed, you may choose to repatriate databases out of business hours.

### Run the repatriation script

Now let's imagine the outage is resolved and run the repatriation script.

1. In the *PowerShell ISE*, in the ...\\Learning Modules\\Business Continuity and Disaster Recovery\\DR-FailoverToReplica\\Demo-FailoverToReplica.ps1 script.
2. Verify that the Catalog Sync process is still running in its PowerShell instance. If necessary, restart it by setting:
  - **\$DemoScenario = 1**, Start synchronizing tenant server, pool, and database configuration info into the catalog
  - Press **F5** to run the script.
3. Then to start the repatriation process, set:
  - **\$DemoScenario = 6**, Repatriate the app into its original region
  - Press **F5** to run the recovery script in a new PowerShell window. Repatriation will take several minutes and can be monitored in the PowerShell window.

```

Windows PowerShell
Refreshing status in 10 seconds...
Databases geo-replicated and failed-over to original region: 50% (2 of 4)

Refreshing status in 10 seconds...
Databases geo-replicated and failed-over to original region: 75% (3 of 4)

Refreshing status in 10 seconds...
Databases geo-replicated and failed-over to original region: 75% (3 of 4)

Refreshing status in 10 seconds...
Databases geo-replicated and failed-over to original region: 100% (4 of 4)
'Wingtip-bgdr2' deployment repatriated into 'northcentralus' region in 3.05 minutes.

```

4. While the script is running, refresh the Events Hub page (<http://events.wingtip-dpt.<user>.trafficmanager.net>)
  - Notice that all the tenants are online and accessible throughout this process.
5. After the repatriation is complete, refresh the Events hub and open the events page for Hawthorn Hall. Notice that this database has been repatriated to the original region.

## Welcome to the Wingtip Tickets Platform!

Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database

search
<b>Venues</b>
Contoso Concert Hall
Dogwood Dojo
Fabrikam Jazz Club
Hawthorn Hall

1 | End


**Wingtip Tickets Platform**

Catalog database: tenantcatalog  
 Server: catalog-dpt-bgdr2  
 Location: northcentralus

Running on Azure SQL Database. © 2018 Microsoft

Learn how to build a SaaS app on SQL database

## Designing the application to ensure app and database are colocated

The application is designed so that it always connects from an instance in the same region as the tenant database. This design reduces latency between the application and the database. This optimization assumes the app-to-database interaction is chattier than the user-to-app interaction.

Tenant databases may be spread across recovery and original regions for some time during repatriation. For each database, the app looks up the region in which the database is located by doing a DNS lookup on the tenant server name. In SQL Database, the server name is an alias. The aliased server name contains the region name. If the application isn't in the same region as the database, it redirects to the instance in the same region as the database server. Redirecting to instance in the same region as the database minimizes latency between app and database.

## Next steps

In this tutorial you learned how to:

- Sync database and elastic pool configuration info into the tenant catalog
- Set up a recovery environment in an alternate region, comprising application, servers, and pools

- Use *geo-replication* to replicate the catalog and tenant databases to the recovery region
- Fail over the application and catalog and tenant databases to the recovery region
- Fail back the application, catalog and tenant databases to the original region after the outage is resolved

You can learn more about the technologies Azure SQL database provides to enable business continuity in the [Business Continuity Overview](#) documentation.

## Additional resources

- [Additional tutorials that build upon the Wingtip SaaS application](#)

# Deploy and explore a sharded multi-tenant application

11/7/2019 • 11 minutes to read • [Edit Online](#)

In this tutorial, you deploy and explore a sample multi-tenant SaaS application that is named Wingtip Tickets. The Wingtip Tickets app is designed to showcase features of Azure SQL Database that simplify the implementation of SaaS scenarios.

This implementation of the Wingtip Tickets app uses a sharded multi-tenant database pattern. The sharding is by tenant identifier. Tenant data is distributed to a particular database according to the tenant identifier values.

This database pattern allows you to store one or more tenants in each shard or database. You can optimize for lowest cost by having each database be shared by multiple tenants. Or you can optimize for isolation by having each database store only one tenant. Your optimization choice can be made independently for each specific tenant. Your choice can be made when the tenant is first stored, or you can change your mind later. The application is designed to work well either way.

## App deploys quickly

The app runs in the Azure cloud and uses Azure SQL Database. The deployment section that follows provides the blue **Deploy to Azure** button. When the button is pressed, the app is fully deployed to your Azure subscription within five minutes. You have full access to work with the individual application components.

The application is deployed with data for three sample tenants. The tenants are stored together in one multi-tenant database.

Anyone can download the C# and PowerShell source code for Wingtip Tickets from [its GitHub repository](#).

## Learn in this tutorial

- How to deploy the Wingtip Tickets SaaS application.
- Where to get the application source code, and management scripts.
- About the servers and databases that make up the app.
- How tenants are mapped to their data with the *catalog*.
- How to provision a new tenant.
- How to monitor tenant activity in the app.

A series of related tutorials is available that build upon this initial deployment. The tutorials explore a range of SaaS design and management patterns. When you work through the tutorials, you are encouraged to step through the provided scripts to see how the different SaaS patterns are implemented.

## Prerequisites

To complete this tutorial, make sure the following prerequisites are completed:

- The latest Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#).

## Deploy the Wingtip Tickets app

### Plan the names

In the steps of this section, you provide a **user** value that is used to ensure resource names are globally unique, and a name for the **resource group** which contains all the resources created by a deployment of the app. For a person named *Ann Finley*, we suggest:

- **User:** **af1** (Their initials, plus a digit. Use a different value (e.g. af2) if you deploy the app a second time.)
- **Resource group:** **wingtip-mt-af1** (wingtip-mt indicates this is the sharded multi-tenant app. Appending the user name af1 correlates the resource group name with the names of the resources it contains.)

Choose your names now, and write them down.

## Steps

1. Click the following blue **Deploy to Azure** button.



2. Enter the required parameter values for the deployment.

### IMPORTANT

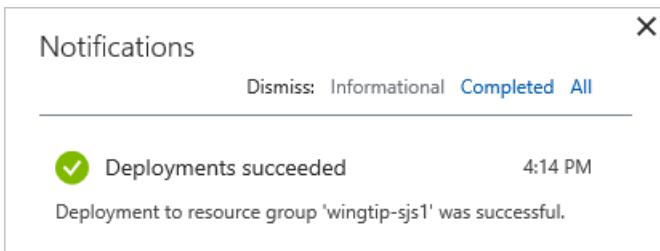
For this demonstration, do not use any pre-existing resource groups, servers, or pools. Instead, choose **Create a new resource group**. Delete this resource group when you are finished with the application to stop related billing. Do not use this application, or any resources it creates, for production. Some aspects of authentication, and the server firewall settings, are intentionally insecure in the app to facilitate the demonstration.

- For **Resource group** - Select **Create new**, and then provide a **Name** for the resource group (case sensitive).
  - Select a **Location** from the drop-down list.
- For **User** - We recommend that you choose a short **User** value.

### 3. Deploy the application.

- Click to agree to the terms and conditions.
- Click **Purchase**.

4. Monitor deployment status by clicking **Notifications**, which is the bell icon to the right of the search box. Deploying the Wingtip app takes approximately five minutes.



## Download and unblock the management scripts

While the application is deploying, download the application source code and management scripts.

### NOTE

Executable contents (scripts, DLLs) may be blocked by Windows when zip files are downloaded from an external source and extracted. When extracting the scripts from a zip file, use the following steps to unblock the .zip file before extracting. By unblocking the .zip file, you ensure the scripts are allowed to run.

1. Browse to [the WingtipTicketsSaaS-MultiTenantDb GitHub repo](#).
2. Click **Clone or download**.
3. Click **Download ZIP** and save the file.
4. Right-click the **WingtipTicketsSaaS-MultiTenantDb-master.zip** file and select **Properties**.
5. On the **General** tab, select **Unblock**, and click **Apply**.
6. Click **OK**.
7. Extract the files.

The scripts are located in the ..\WingtipTicketsSaaS-MultiTenantDb-master\Learning Modules\ folder.

## Update the configuration file for this deployment

Before running any scripts, set the *resource group* and *user* values in **UserConfig.psm1**. Set these variables to the same values you set during deployment.

1. Open ...\\Learning Modules\\UserConfig.psm1 in the *PowerShell ISE*.
2. Update *ResourceGroupName* and *Name* with the specific values for your deployment (on lines 10 and 11 only).
3. Save the changes.

The values set in this file are used by all the scripts, so it is important they are accurate. If you redeploy the app, you must choose different values for User and Resource Group. Then update the UserConfig.psm1 file again with the new values.

## Run the application

In the Wingtip app, the tenants are venues. A venue can be concert hall, a sports club, or any other location that hosts events. The venues register in Wingtip as customers, and a tenant identifier is generated for each venue. Each venue lists its upcoming events in Wingtip, so the public can buy tickets to the events.

Each venue gets a personalized web app to list their events and sell tickets. Each web app is independent and isolated from other tenants. Internally in Azure SQL Database, each the data for each tenant is stored in a sharded multi-tenant database, by default. All data is tagged with the tenant identifier.

A central **Events Hub** webpage provides a list of links to the tenants in your particular deployment. Use the following steps to experience the **Events Hub** webpage and an individual web app:

1. Open the **Events Hub** in your web browser:
  - <http://events.wingtip-mt.<user>.trafficmanager.net> (*Replace <user> with your deployment's user value.*)



## Events Hub

### WELCOME

**Welcome to the Wingtip Tickets Platform!** Explore what's on at the venues below and enjoy secure ticket purchasing in the cloud. Powered by Microsoft Azure, using SaaS technology built on SQL Database.

search

#### Venues

Contoso Concert Hall

Dogwood Dojo

Fabrikam Jazz Club



1 | End



## Wingtip Tickets Platform

Running on Azure SQL Database

LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE

Running on Azure SQL Database. © 2017 Microsoft  
server: catalog\_gmt100wtp.database.windows.net  
database: tenantcatalog

2. Click **Fabrikam Jazz Club** in the Events Hub.

Fabrikam Jazz Club

ALL SESSIONS

WELCOME [SIGN IN]

Next | Jazz Grabbs You

Tickets >

OCT 20 FRI Jazz Grabbs You Tickets

OCT 22 Smokey Sam on Fire Tickets

my|tickets

Update your list of favorites and never miss an event!

Sign In

### Azure Traffic Manager

To control the distribution of incoming requests, the Wingtip app uses [Azure Traffic Manager](#). The events page for

each tenant includes the tenant name in its URL. Each URL also includes your specific User value. Each URL obeys the shown format by using the following steps:

- `http://events.wingtip-mt.<user>.trafficmanager.net/fabrikamjazzclub`
1. The events app parses the tenant name from the URL. The tenant name is *fabrikamjazzclub* in the preceding example URL.
  2. The app then hashes the tenant name to create a key to access a catalog using [shard map management](#).
  3. The app finds the key in the catalog, and obtains the corresponding location of the tenant's database.
  4. The app uses the location info to find and access the one database that contains all the data for the tenant.

## Events Hub

1. The **Events Hub** lists all the tenants that are registered in the catalog, and their venues.
2. The **Events Hub** uses extended metadata in the catalog to retrieve the tenant's name associated with each mapping to construct the URLs.

In a production environment, you typically create a CNAME DNS record to [point a company internet domain](#) to the traffic manager profile.

## Start generating load on the tenant databases

Now that the app is deployed, let's put it to work! The *Demo-LoadGenerator* PowerShell script starts a workload running for each tenant. The real-world load on many SaaS apps is typically sporadic and unpredictable. To simulate this type of load, the generator produces a load distributed across all tenants. The load includes randomized bursts on each tenant occurring at randomized intervals. It takes several minutes for the load pattern to emerge, so it's best to let the generator run for at least three or four minutes before monitoring the load.

1. In the *PowerShell ISE*, open the `..\Learning Modules\Utilities\Demo-LoadGenerator.ps1` script.
2. Press **F5** to run the script and start the load generator (leave the default parameter values for now).

The *Demo-LoadGenerator.ps1* script opens another PowerShell session where the load generator runs. The load generator runs in this session as a foreground task that invokes background load-generation jobs, one for each tenant.

After the foreground task starts, it remains in a job-invoking state. The task starts additional background jobs for any new tenants that are subsequently provisioned.

Closing the PowerShell session stops all jobs.

You might want to restart the load generator session to use different parameter values. If so, close the PowerShell generation session, and then rerun the *Demo-LoadGenerator.ps1*.

## Provision a new tenant into the sharded database

The initial deployment includes three sample tenants in the *Tenants1* database. Let's create another tenant and observe its effects on the deployed application. In this step, you press one key to create a new tenant:

1. Open `..\Learning Modules\Provision and Catalog\Demo-ProvisionTenants.ps1` in the *PowerShell ISE*.
2. Press **F5** (not **F8**) to run the script (leave the default values for now).

### NOTE

You must run the PowerShell scripts only by pressing the **F5** key, not by pressing **F8** to run a selected part of the script. The problem with **F8** is that the `$PSScriptRoot` variable is not evaluated. This variable is needed by many scripts to navigate folders, invoke other scripts, or import modules.

The new Red Maple Racing tenant is added to the *Tenants1* database and registered in the catalog. The new tenant's ticket-selling **Events** site opens in your browser:

A screenshot of a website for "Red Maple Racing". At the top left is a logo with two orange tickets. The top center has the text "Red Maple Racing". On the far right are links for "WELCOME [SIGN IN]". Below the header is a large, blurred image of a race car cockpit from a low angle, showing the driver's helmet and the steering wheel. At the bottom left is a blue button with the text "OCT 21 SAT" and "Event 3" above "Performer 3". At the bottom right is an orange button with the word "Tickets" and a right-pointing arrow. To the right of the main content area is a sidebar with a "my|tickets" section containing the text "Update your list of favorites and never miss an event!".

Refresh the **Events Hub**, and the new tenant now appears in the list.

## Provision a new tenant in its own database

The sharded multi-tenant model allows you to choose whether to provision a new tenant into a database that contains other tenants, or into a database of its own. A tenant isolated in its own database enjoys the following benefits:

- The performance of the tenant's database can be managed without the need to compromise with the needs of other tenants.
- If necessary, the database can be restored to an earlier point in time, because no other tenants would be affected.

You might choose to put free-trial customers, or economy customers, into multi-tenant databases. You could put each premium tenant into its own dedicated database. If you create lots of databases that contain only one tenant, you can manage them all collectively in an elastic pool to optimize resource costs.

Next, we provision another tenant, this time in its own database:

1. In ...\\Learning Modules\\Provision and Catalog\\*Demo-ProvisionTenants.ps1*, modify \$TenantName to **Salix Salsa**, \$VenueType to **dance** and \$Scenario to **2**.
2. Press **F5** to run the script again.
  - This **F5** press provisions the new tenant in a separate database. The database and the tenant are registered in the catalog. Then the browser opens to the Events page of the tenant.

[ALL PERFORMANCES](#)

WELCOME [SIGN IN]



A couple performing salsa dance, woman in red dress.

Next | Event 3 Tickets >

OCT  
21  
SAT

Event 3

Performer 3

[Tickets](#)[my|tickets](#)

Update your list of favorites and  
never miss an event!

- Scroll to the bottom of the page. There in the banner you see the database name in which the tenant data is stored.
3. Refresh the **Events Hub** and the two new tenants now appears in the list.

## Explore the servers and tenant databases

Now we look at some of the resources that were deployed:

1. In the [Azure portal](#), browse to the list of resource groups. Open the resource group you created when you deployed the application.

The screenshot shows the Microsoft Azure portal interface for a resource group named 'wingtip-mt-bga'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, SETTINGS (Quickstart, Resource costs, Deployments, Policies, Properties, Locks, Automation script), and a plus sign icon for creating new resources. The main content area displays a table of 9 items under the 'Essentials' section. The columns are NAME, TYPE, and LOCATION. The items listed are:

NAME	TYPE	LOCATION
catalog-mt-bga	SQL server	South Central US
basetenantdb	SQL database	South Central US
tenantcatalog	SQL database	South Central US
eventsntp-mt-bga	Traffic Manager profile	global
eventsntp-mt-bga	App Service plan	South Central US
eventsntp-mt-bga	App Service	South Central US
tenants1-mt-bga	SQL server	South Central US
salixsalsa	SQL database	South Central US
tenants1	SQL database	South Central US

- Click **catalog-mt<user>** server. The catalog server contains two databases named *tenantcatalog* and *basetenantdb*. The *basetenantdb* database is an empty template database. It is copied to create a new tenant database, whether used for many tenants or just one tenant.

The screenshot shows the Microsoft Azure portal interface for a SQL server named 'catalog-mt-bga'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Quick start, Firewall / Virtual Networks (P..., Failover groups, Long-term backup retention, Auditing & Threat Detection), and a plus sign icon for creating new resources. The main content area displays the properties of the SQL server. Key details include:

- Resource group:** wingtip-mt-bga
- Status:** Available
- Location:** South Central US
- Subscription:** (change)
- Server admin:** developer
- Firewall / Virtual Networks (Preview):** Show firewall settings
- Active Directory admin:** Not configured

The 'SQL databases' section lists two databases:

DATABASE	STATUS	PRICING TIER
basetenantdb	Online	Standard: S0
tenantcatalog	Online	Standard: S0

- Go back to the resource group and select the *tenants1-mt* server that holds the tenant databases.

- The *tenants1* database is a multi-tenant database in which the original three tenants, plus the first tenant you added, are stored. It is configured as a 50 DTU Standard database.
- The **salixsalsa** database holds the Salix Salsa dance venue as its only tenant. It is configured as a Standard edition database with 50 DTUs by default.

## Monitor the performance of the database

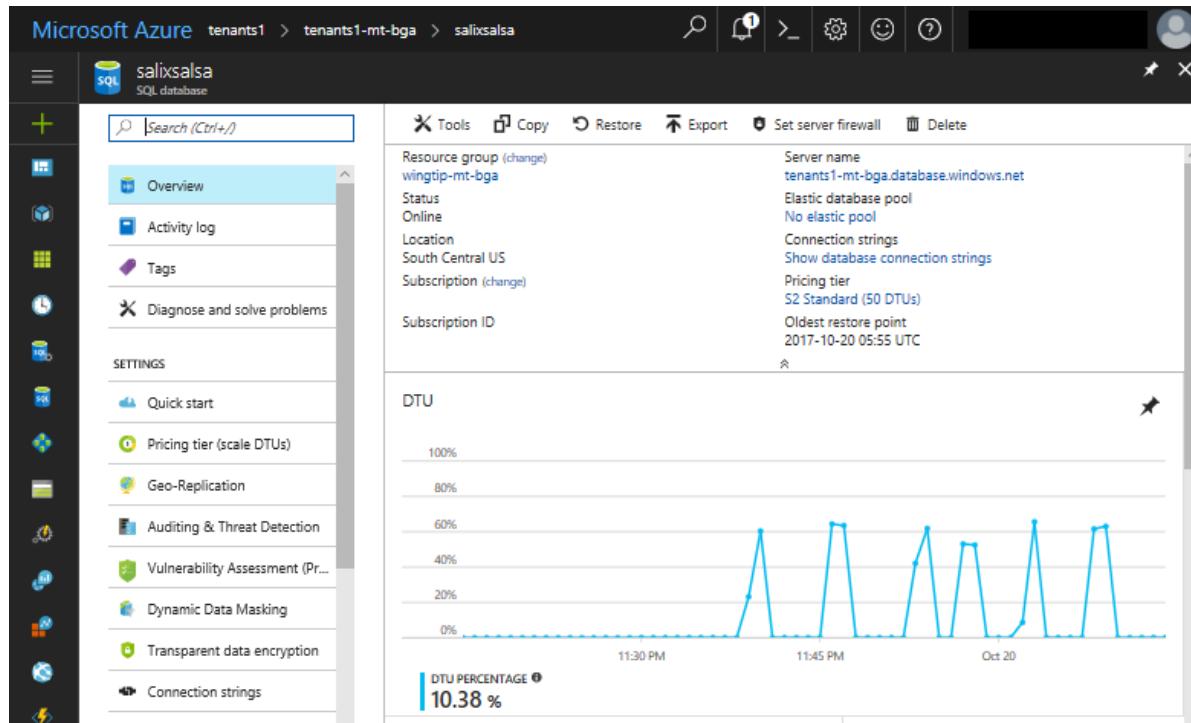
If the load generator has been running for several minutes, enough telemetry is available to look at the database monitoring capabilities built into the Azure portal.

1. Browse to the **tenants1-<user>** server, and click **tenants1** to view resource utilization for the database that has four tenants in it. Each tenant is subject to a sporadic heavy load from the load generator:

The DTU utilization chart nicely illustrates how a multi-tenant database can support an unpredictable workload across many tenants. In this case, the load generator is applying a sporadic load of roughly 30 DTUs to each tenant. This load equates to 60% utilization of a 50 DTU database. Peaks that exceed 60% are

the result of load being applied to more than one tenant at the same time.

2. Browse to the **tenants1-mt<user>** server, and click the **salixsalsa** database. You can see the resource utilization on this database that contains only one tenant.



The load generator is applying a similar load to each tenant, regardless of which database each tenant is in. With only one tenant in the **salixsalsa** database, you can see that the database could sustain a much higher load than the database with several tenants.

### Resource allocations vary by workload

Sometimes a multi-tenant database requires more resources for good performance than does a single-tenant database, but not always. The optimal allocation of resources depends on the particular workload characteristics for the tenants in your system.

The workloads generated by the load generator script are for illustration purposes only.

## Additional resources

- To learn about multi-tenant SaaS applications, see [Design patterns for multi-tenant SaaS applications](#).
- To learn about elastic pools, see:
  - [Elastic pools help you manage and scale multiple Azure SQL databases](#)
  - [Scaling out with Azure SQL Database](#)

## Next steps

In this tutorial you learned:

- How to deploy the Wingtip Tickets SaaS Multi-tenant Database application.
- About the servers, and databases that make up the app.
- Tenants are mapped to their data with the *catalog*.
- How to provision new tenants, into a multi-tenant database and single-tenant database.
- How to view pool utilization to monitor tenant activity.
- How to delete sample resources to stop related billing.

Now try the [Provision and catalog tutorial](#).

# Provision and catalog new tenants in a SaaS application using a sharded multi-tenant Azure SQL database

11/15/2019 • 12 minutes to read • [Edit Online](#)

This article covers the provisioning and cataloging of new tenants, in a *multi-tenant sharded database* model or pattern.

This article has two major parts:

- [Conceptual discussion](#) of the provisioning and cataloging of new tenants.
- [Tutorial](#) that highlights the PowerShell script code that accomplishes the provisioning and cataloging.
  - The tutorial uses the Wingtip Tickets SaaS application, adapted to the multi-tenant sharded database pattern.

## Database pattern

This section, plus a few more that follow, discuss the concepts of the multi-tenant sharded database pattern.

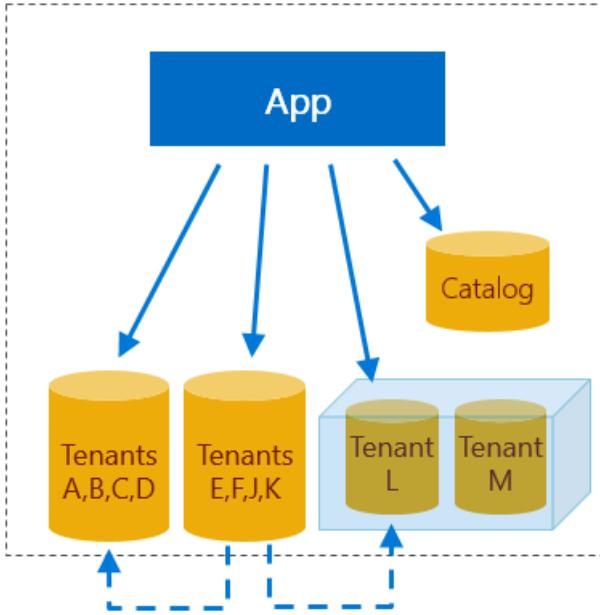
In this multi-tenant sharded model, the table schemas inside each database include a tenant key in the primary key of tables that store tenant data. The tenant key enables each individual database to store 0, 1, or many tenants. The use of sharded databases makes it easy for the application system to support a very large number of tenants. All the data for any one tenant is stored in one database. The large number of tenants are distributed across the many sharded databases. A catalog database stores the mapping of each tenant to its database.

### **Isolation versus lower cost**

A tenant that has a database all to itself enjoys the benefits of isolation. The tenant can have the database restored to an earlier date without being restricted by the impact on other tenants. Database performance can be tuned to optimize for the one tenant, again without having to compromise with other tenants. The problem is that isolation costs more than it costs to share a database with other tenants.

When a new tenant is provisioned, it can share a database with other tenants, or it can be placed into its own new database. Later you can change your mind and move the database to the other situation.

Databases with multiple tenants and single tenants are mixed in the same SaaS application, to optimize cost or isolation for each tenant.



## Tenant catalog pattern

When you have two or more databases that each contain at least one tenant, the application must have a way to discover which database stores the tenant of current interest. A catalog database stores this mapping.

### Tenant key

For each tenant, the Wingtip application can derive a unique key, which is the tenant key. The app extracts the tenant name from the webpage URL. The app hashes the name to obtain the key. The app uses the key to access the catalog. The catalog cross-references information about the database in which the tenant is stored. The app uses the database info to connect. Other tenant key schemes can also be used.

Using a catalog allows the name or location of a tenant database to be changed after provisioning without disrupting the application. In a multi-tenant database model, the catalog accommodates moving a tenant between databases.

### Tenant metadata beyond location

The catalog can also indicate whether a tenant is offline for maintenance or other actions. And the catalog can be extended to store additional tenant or database metadata, such as the following items:

- The service tier or edition of a database.
- The version of the database schema.
- The tenant name and its SLA (service level agreement).
- Information to enable application management, customer support, or devops processes.

The catalog can also be used to enable cross-tenant reporting, schema management, and data extract for analytics purposes.

### Elastic Database Client Library

In Wingtip, the catalog is implemented in the *tenantcatalog* database. The *tenantcatalog* is created using the Shard Management features of the [Elastic Database Client Library \(EDCL\)](#). The library enables an application to create, manage, and use a *shard map* that is stored in a database. A shard map cross-references the tenant key with its shard, meaning its sharded database.

During tenant provisioning, EDCL functions can be used from applications or PowerShell scripts to create the entries in the shard map. Later the EDCL functions can be used to connect to the correct database. The EDCL caches connection information to minimize the traffic on the catalog database and speed up the process of connecting.

## IMPORTANT

Do *not* edit the data in the catalog database through direct access! Direct updates are not supported due to the high risk of data corruption. Instead, edit the mapping data by using EDCL APIs only.

# Tenant provisioning pattern

## Checklist

When you want to provision a new tenant into an existing shared database, of the shared database you must ask the following questions:

- Does it have enough space left for the new tenant?
- Does it have tables with the necessary reference data for the new tenant, or can the data be added?
- Does it have the appropriate variation of the base schema for the new tenant?
- Is it in the appropriate geographic location close to the new tenant?
- Is it at the right service tier for the new tenant?

When you want the new tenant to be isolated in its own database, you can create it to meet the specifications for the tenant.

After the provisioning is complete, you must register the tenant in the catalog. Finally, the tenant mapping can be added to reference the appropriate shard.

## Template database

Provision the database by executing SQL scripts, deploying a bacpac, or copying a template database. The Wingtip apps copy a template database to create new tenant databases.

Like any application, Wingtip will evolve over time. At times, Wingtip will require changes to the database.

Changes may include the following items:

- New or changed schema.
- New or changed reference data.
- Routine database maintenance tasks to ensure optimal app performance.

With a SaaS application, these changes need to be deployed in a coordinated manner across a potentially massive fleet of tenant databases. For these changes to be in future tenant databases, they need to be incorporated into the provisioning process. This challenge is explored further in the [schema management tutorial](#).

## Scripts

The tenant provisioning scripts in this tutorial support both of the following scenarios:

- Provisioning a tenant into an existing database shared with other tenants.
- Provisioning a tenant into its own database.

Tenant data is then initialized and registered in the catalog shard map. In the sample app, databases that contain multiple tenants are given a generic name, such as *tenants1* or *tenants2*. Databases that contain a single tenant are given the tenant's name. The specific naming conventions used in the sample are not a critical part of the pattern, as the use of a catalog allows any name to be assigned to the database.

# Tutorial begins

In this tutorial, you learn how to:

- Provision a tenant into a multi-tenant database
- Provision a tenant into a single-tenant database

- Provision a batch of tenants into both multi-tenant and single-tenant databases
- Register a database and tenant mapping in a catalog

#### Prerequisites

To complete this tutorial, make sure the following prerequisites are completed:

- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- The Wingtip Tickets SaaS Multi-tenant Database app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Multi-tenant Database application](#)
- Get the Wingtip scripts and source code:
  - The Wingtip Tickets SaaS Multi-tenant Database scripts and application source code are available in the [WingtipTicketsSaaS-MultitenantDB](#) GitHub repo.
  - See the [general guidance](#) for steps to download and unblock the Wingtip scripts.

## Provision a tenant into a database *shared* with other tenants

In this section, you see a list of the major actions for provisioning that are taken by the PowerShell scripts. Then you use the PowerShell ISE debugger to step through the scripts to see the actions in code.

#### Major actions of provisioning

The following are key elements of the provisioning workflow you step through:

- **Calculate the new tenant key:** A hash function is used to create the tenant key from the tenant name.
- **Check if the tenant key already exists:** The catalog is checked to ensure the key has not already been registered.
- **Initialize tenant in the default tenant database:** The tenant database is updated to add the new tenant information.
- **Register tenant in the catalog:** The mapping between the new tenant key and the existing tenants1 database is added to the catalog.
- **Add the tenant's name to a catalog extension table:** The venue name is added to the Tenants table in the catalog. This addition shows how the Catalog database can be extended to support additional application-specific data.
- **Open Events page for the new tenant:** The *Bushwillow Blues* events page is opened in the browser.

#### Debugger steps

To understand how the Wingtip app implements new tenant provisioning in a shared database, add a breakpoint and step through the workflow:

1. In the *PowerShell ISE*, open ...\\Learning Modules\\ProvisionTenants\\*Demo-ProvisionTenants.ps1* and set the following parameters:
  - **\$TenantName = Bushwillow Blues**, the name of a new venue.
  - **\$VenueType = blues**, one of the pre-defined venue types: blues, classicalmusic, dance, jazz, judo, motorracing, multipurpose, opera, rockmusic, soccer (lowercase, no spaces).
  - **\$DemoScenario = 1**, to provision a tenant in a shared database with other tenants.
2. Add a breakpoint by putting your cursor anywhere on line 38, the line that says: *New-Tenant* ` , and then press **F9**.

```

35 ### Provision a tenant in a shared database with other tenants
36 if ($Scenario -eq 1)
37 {
38 New-Tenant
39 -TenantName $TenantName
40 -VenueType $VenueType
41 -PostalCode $PostalCode
42 -ErrorAction Stop
43 > $null
44
45 Write-Output "Provisioning complete for tenant '$TenantName'"
46
47 # Open the events page for the new venue
48 Start-Process "http://events.wingtip-mt.$($wtpUser.Name).trafficman
49

```

3. Run the script by pressing **F5**.
4. After script execution stops at the breakpoint, press **F11** to step into the code.

A screenshot of the Windows PowerShell ISE interface. The title bar says "Windows PowerShell ISE". The menu bar has "File", "Edit", "View", "Tools", "Debug", "Add-ons", and "Help". The "Debug" menu is open, showing options like "Step Over" (F10), "Step Into" (F11, highlighted in blue), "Step Out" (Shift+F11), "Run/Continue" (F5), "Break All" (Ctrl+B), "Stop Debugger" (Shift+F5), "Toggle Breakpoint" (F9), "Remove All Breakpoints" (Ctrl+Shift+F9), "Enable All Breakpoints", "Disable All Breakpoints", "List Breakpoints" (Ctrl+Shift+L), and "Display Call Stack" (Ctrl+Shift+D). The main code editor window shows a PowerShell script named "Demo-ProvisionTenants.ps1". The script contains several lines of PowerShell code, including variable assignments and a conditional block. A specific line of code at the bottom is highlighted in red: "Start-Process "http://events.wingtip-mt.\$(\$wtpUser.Name).trafficmanager.net/api/events?tenant=\$TenantName&id=\$EventId&start=2018-01-01T00:00:00&end=2018-01-01T01:00:00&count=10&sort=desc&filter=category eq 'New'".

```

30 Initial
31
32 $wtpUser
33 $config
34
35 ##### Provisioning
36 if ($Scopes -eq "New")
37 {
38 New-Tenant -Name $TenantName -Key $TenantKey
39
40 Write-Output "Provisioning complete for tenant '$TenantName'"
41
42 # Open the events page for the new venue
43 Start-Process "http://events.wingtip-mt.$($wtpUser.Name).trafficmanager.net/api/events?tenant=$TenantName&id=$EventId&start=2018-01-01T00:00:00&end=2018-01-01T01:00:00&count=10&sort=desc&filter=category eq 'New'"

```

- Trace the script's execution using the **Debug** menu options, **F10** and **F11**, to step over or into called functions.

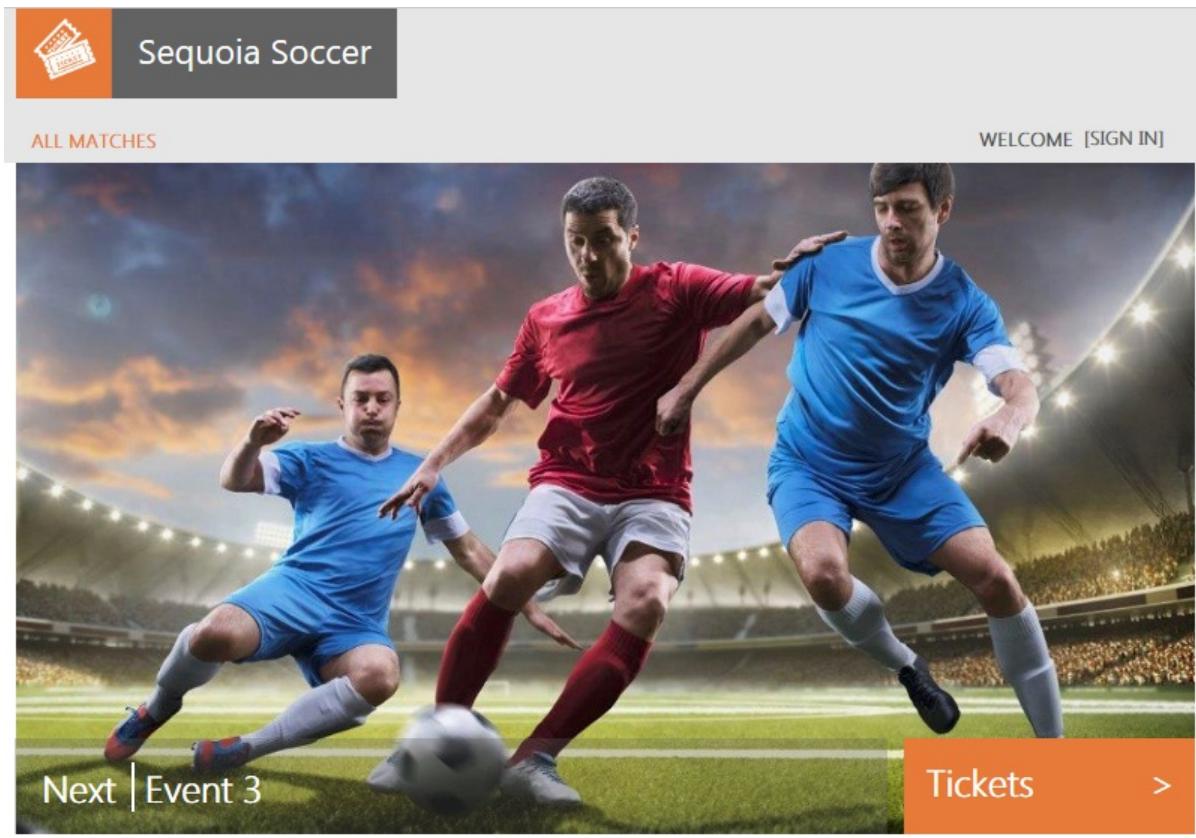
For more information about debugging PowerShell scripts, see [Tips on working with and debugging PowerShell scripts](#).

## Provision a tenant in its own database

### Major actions of provisioning

The following are key elements of the workflow you step through while tracing the script:

- Calculate the new tenant key:** A hash function is used to create the tenant key from the tenant name.
- Check if the tenant key already exists:** The catalog is checked to ensure the key has not already been registered.
- Create a new tenant database:** The database is created by copying the *basetenantdb* database using a Resource Manager template. The new database name is based on the tenant's name.
- Add database to catalog:** The new tenant database is registered as a shard in the catalog.
- Initialize tenant in the default tenant database:** The tenant database is updated to add the new tenant information.
- Register tenant in the catalog:** The mapping between the new tenant key and the *sequoiasoccer* database is added to the catalog.
- Tenant name is added to the catalog:** The venue name is added to the Tenants extension table in the catalog.
- Open Events page for the new tenant:** The *Sequoia Soccer* Events page is opened in the browser.



NOV  
16  
THU

**Event 3**

Performer 3

Tickets

**my|tickets**

Update your list of favorites and never miss an event!

#### Debugger steps

Now walk through the script process when creating a tenant in its own database:

1. Still in ...\\Learning Modules\\ProvisionTenants\\*Demo-ProvisionTenants.ps1* set the following parameters:
  - **\$TenantName = Sequoia Soccer**, the name of a new venue.
  - **\$VenueType = soccer**, one of the pre-defined venue types: blues, classicalmusic, dance, jazz, judo, motorracing, multipurpose, opera, rockmusic, soccer (lower case, no spaces).
  - **\$DemoScenario = 2**, to provision a tenant into its own database.
2. Add a new breakpoint by putting your cursor anywhere on line 57, the line that says: & \$PSScriptRoot\\New-TenantAndDatabase ` , and press **F9**.

```

54 ### Provision a tenant in its own database
55 if ($Scenario -eq 2)
56 {
57 & $PSScriptRoot\New-TenantAndDatabase
58 -TenantName $TenantName
59 -VenueType $VenueType
60 -PostalCode $PostalCode
61 -ErrorAction Stop
62 > $null
63

```

3. Run the script by pressing **F5**.
4. After the script execution stops at the breakpoint, press **F11** to step into the code. Use **F10** and **F11** to step over and step into functions to trace the execution.

## Provision a batch of tenants

This exercise provisions a batch of 17 tenants. It's recommended you provision this batch of tenants before starting other Wingtip Tickets tutorials so there are more databases to work with.

1. In the *PowerShell ISE*, open ...\\Learning Modules\\ProvisionTenants\\*Demo-ProvisionTenants.ps1* and change the \$DemoScenario parameter to 4:

- \$DemoScenario = 4, to provision a batch of tenants into a shared database.

2. Press **F5** and run the script.

### Verify the deployed set of tenants

At this stage, you have a mix of tenants deployed into a shared database and tenants deployed into their own databases. The Azure portal can be used to inspect the databases created. In the [Azure portal](#), open the **tenants1-mt-<USER>** server by browsing to the list of SQL servers. The **SQL databases** list should include the shared **tenants1** database and the databases for the tenants that are in their own database:

The screenshot shows the Azure portal interface for a SQL server named "tenants1-mt-bgb". The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Firewall / Virtual Networks (Preview), Failover groups, Long-term backup retention, Auditing & Threat Detection, Transparent data encryption, and Active Directory admin. The main content area displays the server's overview, including its resource group (wingtip-mt-bgb), status (Available), location (South Central US), subscription (Standard), and a link to view recommended elastic database pool settings. Below this, the "SQL databases" section is highlighted with a red border. It lists three databases: "salixsalsa" (Status: Online, Pricing Tier: Standard: S2), "sequoiasoccer" (Status: Online, Pricing Tier: Standard: S2), and "tenants1" (Status: Online, Pricing Tier: Standard: S2).

DATABASE	STATUS	PRICING TIER
salixsalsa	Online	Standard: S2
sequoiasoccer	Online	Standard: S2
tenants1	Online	Standard: S2

While the Azure portal shows the tenant databases, it doesn't let you see the tenants *inside* the shared database. The full list of tenants can be seen in the **Events Hub** webpage of Wingtip, and by browsing the catalog.

### Using Wingtip Tickets events hub page

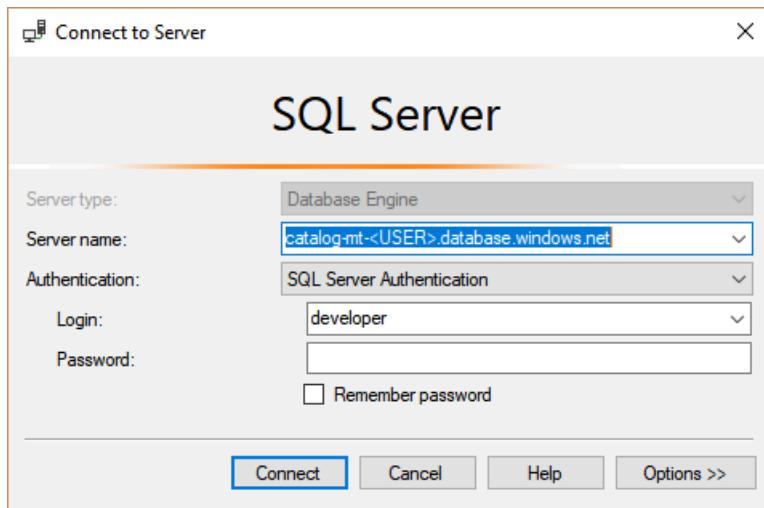
Open the Events Hub page in the browser (<http://events.wingtip-mt.<USER>.trafficmanager.net>)

### Using catalog database

The full list of tenants and the corresponding database for each is available in the catalog. A SQL view is provided that joins the tenant name to the database name. The view nicely demonstrates the value of extending the metadata that is stored in the catalog.

- The SQL view is available in the tenantcatalog database.
- The tenant name is stored in the Tenants table.
- The database name is stored in the Shard Management tables.

1. In SQL Server Management Studio (SSMS), connect to the tenants server at **catalog-mt-<USER>.database.windows.net**, with Login = **developer**, and Password = **P@ssword1**



2. In the SSMS Object Explorer, browse to the views in the *tenantcatalog* database.
3. Right click on the view *TenantsExtended* and choose **Select Top 1000 Rows**. Note the mapping between tenant name and database for the different tenants.

	TenantId	TenantName	ServicePlan	ServerName	DatabaseName
1	0x4D203394	Salix Salsa	standard	tenants1-mt-bgb.database.windows.net	salixsalsa
2	0x58BB0FA9	Sequoia Soccer	standard	tenants1-mt-bgb.database.windows.net	sequoiasoccer
3	0x04330A60	Tamarind Studio	standard	tenants1-mt-bgb.database.windows.net	tenants1
4	0x063922B4	Lime Tree Track	standard	tenants1-mt-bgb.database.windows.net	tenants1
5	0x076E64E2	Papaya Players	standard	tenants1-mt-bgb.database.windows.net	tenants1
6	0x0FABEA35	Blue Oak Jazz Club	standard	tenants1-mt-bgb.database.windows.net	tenants1
7	0x254AA9D2	Osage Opera	standard	tenants1-mt-bgb.database.windows.net	tenants1
8	0x2E6F8C1F	Dogwood Dojo	Standard	tenants1-mt-bgb.database.windows.net	tenants1

## Other provisioning patterns

This section discusses other interesting provisioning patterns.

### Pre-provisioning databases in elastic pools

The pre-provisioning pattern exploits the fact that when using elastic pools, billing is for the pool not the databases. Thus databases can be added to an elastic pool before they are needed without incurring extra cost. This pre-provisioning significantly reduces the time taken to provision a tenant into a database. The number of databases pre-provisioned can be adjusted as needed to keep a buffer suitable for the anticipated provisioning rate.

### Auto-provisioning

In the auto-provisioning pattern, a dedicated provisioning service is used to provision servers, pools, and databases automatically as needed. This automation includes the pre-provisioning of databases in elastic pools. And if databases are decommissioned and deleted, the gaps this creates in elastic pools can be filled by the provisioning service as desired.

This type of automated service could be simple or complex. For example, the automation could handle provisioning across multiple geographies, and could set up geo-replication for disaster recovery. With the auto-provisioning pattern, a client application or script would submit a provisioning request to a queue to be processed by a provisioning service. The script would then poll to detect completion. If pre-provisioning is used, requests

would be handled quickly, while a background service would manage the provisioning of a replacement database.

## Additional resources

- [Elastic database client library](#)
- [How to Debug Scripts in Windows PowerShell ISE](#)

## Next steps

In this tutorial you learned how to:

- Provision a single new tenant into a shared multi-tenant database and its own database
- Provision a batch of additional tenants
- Step through the details of provisioning tenants, and registering them into the catalog

Try the [Performance monitoring tutorial](#).

# Monitor and manage performance of sharded multi-tenant Azure SQL database in a multi-tenant SaaS app

11/7/2019 • 10 minutes to read • [Edit Online](#)

In this tutorial, several key performance management scenarios used in SaaS applications are explored. Using a load generator to simulate activity across sharded multi-tenant databases, the built-in monitoring and alerting features of SQL Database are demonstrated.

The Wingtip Tickets SaaS Multi-tenant Database app uses a sharded multi-tenant data model, where venue (tenant) data is distributed by tenant ID across potentially multiple databases. Like many SaaS applications, the anticipated tenant workload pattern is unpredictable and sporadic. In other words, ticket sales may occur at any time. To take advantage of this typical database usage pattern, databases can be scaled up and down to optimize the cost of a solution. With this type of pattern, it's important to monitor database resource usage to ensure that loads are reasonably balanced across potentially multiple databases. You also need to ensure that individual databases have adequate resources and are not hitting their [DTU](#) limits. This tutorial explores ways to monitor and manage databases, and how to take corrective action in response to variations in workload.

In this tutorial you learn how to:

- Simulate usage on a sharded multi-tenant database by running a provided load generator
- Monitor the database as it responds to the increase in load
- Scale up the database in response to the increased database load
- Provision a tenant into a single-tenant database

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS Multi-tenant Database app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Multi-tenant Database application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)

## Introduction to SaaS performance management patterns

Managing database performance consists of compiling and analyzing performance data, and then reacting to this data by adjusting parameters to maintain an acceptable response time for your application.

### Performance management strategies

- To avoid having to manually monitor performance, it's most effective to **set alerts that trigger when databases stray out of normal ranges**.
- To respond to short-term fluctuations in the compute size of a database, the **DTU level can be scaled up or down**. If this fluctuation occurs on a regular or predictable basis, **scaling the database can be scheduled to occur automatically**. For example, scale down when you know your workload is light, maybe overnight, or during weekends.
- To respond to longer-term fluctuations, or changes in the tenants, **individual tenants can be moved into other database**.
- To respond to short-term increases in *individual* tenant load, **individual tenants can be taken out of a database and assigned an individual compute size**. Once the load is reduced, the tenant can then be returned to the multi-tenant database. When this is known in advance, tenants can be moved pre-emptively to ensure the database always has the resources it needs, and to avoid impact on other tenants in the multi-tenant

database. If this requirement is predictable, such as a venue experiencing a rush of ticket sales for a popular event, then this management behavior can be integrated into the application.

The [Azure portal](#) provides built-in monitoring and alerting on most resources. For SQL Database, monitoring and alerting is available on databases. This built-in monitoring and alerting is resource-specific, so it's convenient to use for small numbers of resources, but is not convenient when working with many resources.

For high-volume scenarios, where you're working with many resources, [Azure Monitor logs](#) can be used. This is a separate Azure service that provides analytics over emitted diagnostic logs and telemetry gathered in a Log Analytics workspace. Azure Monitor logs can collect telemetry from many services and be used to query and set alerts.

## Get the Wingtip Tickets SaaS Multi-tenant Database application source code and scripts

The Wingtip Tickets SaaS Multi-tenant Database scripts and application source code are available in the [WingtipTicketsSaaS-MultitenantDB](#) GitHub repo. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

## Provision additional tenants

For a good understanding of how performance monitoring and management works at scale, this tutorial requires you to have multiple tenants in a sharded multi-tenant database.

If you have already provisioned a batch of tenants in a prior tutorial, skip to the [Simulate usage on all tenant databases](#) section.

1. In the **PowerShell ISE**, open ...\\Learning Modules\\Performance Monitoring and Management\\*Demo-PerformanceMonitoringAndManagement.ps1*. Keep this script open as you'll run several scenarios during this tutorial.
2. Set **\$DemoScenario = 1**, *Provision a batch of tenants*
3. Press **F5** to run the script.

The script deploys 17 tenants into the multi-tenant database in a few minutes.

The *New-TenantBatch* script creates new tenants with unique tenant keys within the sharded multi-tenant database and initializes them with the tenant name and venue type. This is consistent with the way the app provisions a new tenant.

## Simulate usage on all tenant databases

The *Demo-PerformanceMonitoringAndManagement.ps1* script is provided that simulates a workload running against the multi-tenant database. The load is generated using one of the available load scenarios:

DEMO	SCENARIO
2	Generate normal intensity load (approximately 30 DTU)
3	Generate load with longer bursts per tenant
4	Generate load with higher DTU bursts per tenant (approximately 70 DTU)
5	Generate a high intensity (approximately 90 DTU) on a single tenant plus a normal intensity load on all other tenants

The load generator applies a *synthetic* CPU-only load to every tenant database. The generator starts a job for each tenant database, which calls a stored procedure periodically that generates the load. The load levels (in DTUs), duration, and intervals are varied across all databases, simulating unpredictable tenant activity.

1. In the **PowerShell ISE**, open ...\\Learning Modules\\Performance Monitoring and Management\\*Demo-PerformanceMonitoringAndManagement.ps1*. Keep this script open as you'll run several scenarios during this tutorial.
2. Set **\$DemoScenario = 2**, *Generate normal intensity load*
3. Press **F5** to apply a load to all your tenants.

Wingtip Tickets SaaS Multi-tenant Database is a SaaS app, and the real-world load on a SaaS app is typically sporadic and unpredictable. To simulate this, the load generator produces a randomized load distributed across all tenants. Several minutes are needed for the load pattern to emerge, so run the load generator for 3-5 minutes before attempting to monitor the load in the following sections.

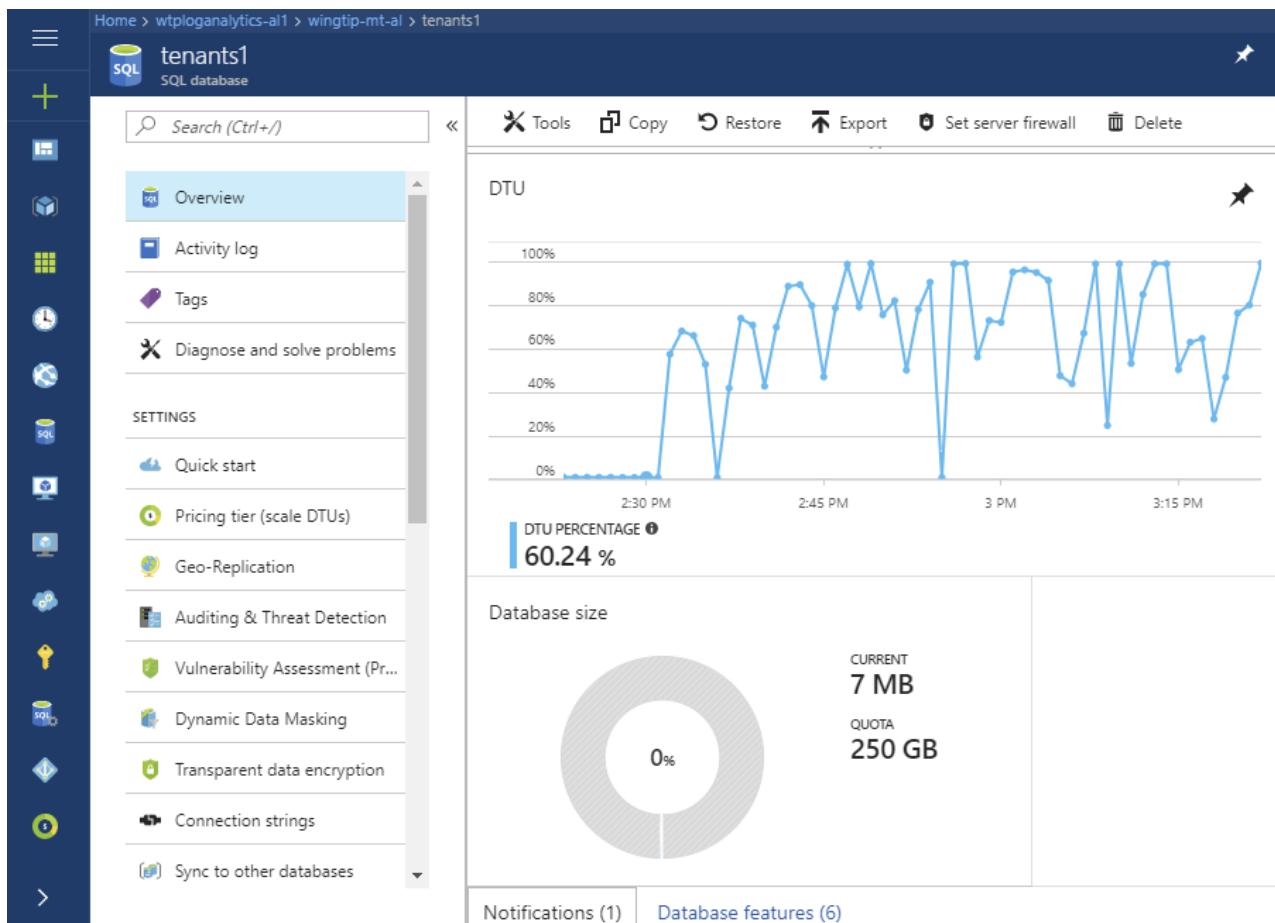
#### IMPORTANT

The load generator is running as a series of jobs in a new PowerShell window. If you close the session, the load generator stops. The load generator remains in a *job-invoking* state where it generates load on any new tenants that are provisioned after the generator is started. Use **Ctrl-C** to stop invoking new jobs and exit the script. The load generator will continue to run, but only on existing tenants.

## Monitor resource usage using the Azure portal

To monitor the resource usage that results from the load being applied, open the portal to the multi-tenant database, **tenants1**, containing the tenants:

1. Open the [Azure portal](#) and browse to the server *tenants1-mt-<USER>*.
2. Scroll down and locate databases and click **tenants1**. This sharded multi-tenant database contains all the tenants created so far.



Observe the **DTU** chart.

## Set performance alerts on the database

Set an alert on the database that triggers on >75% utilization as follows:

1. Open the `tenants1` database (on the `tenants1-mt-<USER>` server) in the [Azure portal](#).
2. Click **Alert Rules**, and then click **+ Add alert**:

Microsoft Azure

Home > tenants1 - Alert rules

**tenants1 - Alert rules**  
SQL database

**Add alert**

Search (Ctrl+ /)

Overview

Activity log

Tags

Diagnose and solve problems

SETTINGS

Quick start

Pricing tier (scale DTUs)

Geo-Replication

Auditing & Threat Detection

Vulnerability Assessment (Pr...)

Dynamic Data Masking

Transparent data encryption

Connection strings

Sync to other databases

Properties

Locks

Automation script

MONITORING

**Alert rules**

You haven't created any alert rules.

The screenshot shows the Microsoft Azure portal interface for managing alert rules. On the left, there's a vertical sidebar with various icons representing different services. The main area is titled 'tenants1 - Alert rules' and shows a list of settings like 'Quick start', 'Pricing tier (scale DTUs)', and 'Auditing & Threat Detection'. A red box highlights the 'Alert rules' item in the 'MONITORING' section at the bottom of the list. Another red box highlights the 'Add alert' button at the top right of the page.

3. Provide a name, such as **High DTU**,

4. Set the following values:

- **Metric = DTU percentage**
- **Condition = greater than**
- **Threshold = 75.**
- **Period = Over the last 30 minutes**

5. Add an email address to the *Additional administrator email(s)* box and click **OK**.

The screenshot shows the Microsoft Azure portal's 'Add an alert rule' interface. The left sidebar lists various services like Storage, Functions, and Logic Apps. The main form is titled 'Add an alert rule' and includes the following fields:

- Resource:** tenants1-ml-al2/tenants1 (servers/dat...)
- Name:** High DTU
- Description:** Description
- Metric:** DTU percentage
- Condition:** greater than
- Threshold:** 75 %
- Period:** Over the last 30 minutes
- Email owners, contributors, and readers:** (checkbox)

A chart below the metric selection shows DTU percentage over time, with a sharp spike reaching 100% around 12 PM.

## Scale up a busy database

If the load level increases on a database to the point that it maxes out the database and reaches 100% DTU usage, then database performance is affected, potentially slowing query response times.

**Short term**, consider scaling up the database to provide additional resources, or removing tenants from the multi-tenant database (moving them out of the multi-tenant database to a stand-alone database).

**Longer term**, consider optimizing queries or index usage to improve database performance. Depending on the application's sensitivity to performance issues its best practice to scale a database up before it reaches 100% DTU usage. Use an alert to warn you in advance.

You can simulate a busy database by increasing the load produced by the generator. Causing the tenants to burst more frequently, and for longer, increasing the load on the multi-tenant database without changing the requirements of the individual tenants. Scaling up the database is easily done in the portal or from PowerShell. This exercise uses the portal.

1. Set \$DemoScenario = 3, Generate load with longer and more frequent bursts per database to increase the intensity of the aggregate load on the database without changing the peak load required by each tenant.

2. Press **F5** to apply a load to all your tenant databases.
3. Go to the **tenants1** database in the Azure portal.

Monitor the increased database DTU usage on the upper chart. It takes a few minutes for the new higher load to kick in, but you should quickly see the database start to hit max utilization, and as the load steadies into the new pattern, it rapidly overloads the database.

1. To scale up the database, click **Pricing tier (scale DTUs)** in the settings blade.
2. Adjust the **DTU** setting to **100**.
3. Click **Apply** to submit the request to scale the database.

Go back to **tenants1 > Overview** to view the monitoring charts. Monitor the effect of providing the database with more resources (although with few tenants and a randomized load it's not always easy to see conclusively until you run for some time). While you are looking at the charts bear in mind that 100% on the upper chart now represents 100 DTUs, while on the lower chart 100% is still 50 DTUs.

Databases remain online and fully available throughout the process. Application code should always be written to retry dropped connections, and so will reconnect to the database.

## Provision a new tenant in its own database

The sharded multi-tenant model allows you to choose whether to provision a new tenant in a multi-tenant database alongside other tenants, or to provision the tenant in a database of its own. By provisioning a tenant in its own database, it benefits from the isolation inherent in the separate database, allowing you to manage the performance of that tenant independently of others, restore that tenant independently of others, etc. For example, you might choose to put free-trial or regular customers in a multi-tenant database, and premium customers in individual databases. If isolated single-tenant databases are created, they can still be managed collectively in an elastic pool to optimize resource costs.

If you already provisioned a new tenant in its own database, skip the next few steps.

1. In the **PowerShell ISE**, open ...\\Learning Modules\\ProvisionTenants\\Demo-ProvisionTenants.ps1.
2. Modify **\$TenantName = "Salix Salsa"** and **\$VenueType = "dance"**
3. Set **\$Scenario = 2**, *Provision a tenant in a new single-tenant database*
4. Press **F5** to run the script.

The script will provision this tenant in a separate database, register the database and the tenant with the catalog, and then open the tenant's Events page in the browser. Refresh the Events Hub page and you will see "Salix Salsa" has been added as a venue.

## Manage performance of an individual database

If a single tenant within a multi-tenant database experiences a sustained high load, it may tend to dominate the database resources and impact other tenants in the same database. If the activity is likely to continue for some time, the tenant can be temporarily moved out of the database and into its own single-tenant database. This allows the tenant to have the extra resources it needs, and fully isolates it from the other tenants.

This exercise simulates the effect of Salix Salsa experiencing a high load when tickets go on sale for a popular event.

1. Open the ...\\Demo-PerformanceMonitoringAndManagement.ps1 script.
2. Set **\$DemoScenario = 5**, *Generate a normal load plus a high load on a single tenant (approximately 90 DTU).*
3. Set **\$SingleTenantName = Salix Salsa**
4. Execute the script using **F5**.

Go to portal and navigate to **salixsalsa** > **Overview** to view the monitoring charts.

## Other performance management patterns

### Tenant self-service scaling

Because scaling is a task easily called via the management API, you can easily build the ability to scale tenant databases into your tenant-facing application, and offer it as a feature of your SaaS service. For example, let tenants self-administer scaling up and down, perhaps linked directly to their billing!

### Scaling a database up and down on a schedule to match usage patterns

Where aggregate tenant usage follows predictable usage patterns, you can use Azure Automation to scale a database up and down on a schedule. For example, scale a database down after 6pm and up again before 6am on weekdays when you know there is a drop in resource requirements.

## Next steps

In this tutorial you learn how to:

- Simulate usage on a sharded multi-tenant database by running a provided load generator
- Monitor the database as it responds to the increase in load
- Scale up the database in response to the increased database load
- Provision a tenant into a single-tenant database

## Additional resources

- [Azure automation](#)

# Run ad hoc analytics queries across multiple Azure SQL databases

11/7/2019 • 7 minutes to read • [Edit Online](#)

In this tutorial, you run distributed queries across the entire set of tenant databases to enable ad hoc interactive reporting. These queries can extract insights buried in the day-to-day operational data of the Wingtip Tickets SaaS app. To do these extractions, you deploy an additional analytics database to the catalog server and use Elastic Query to enable distributed queries.

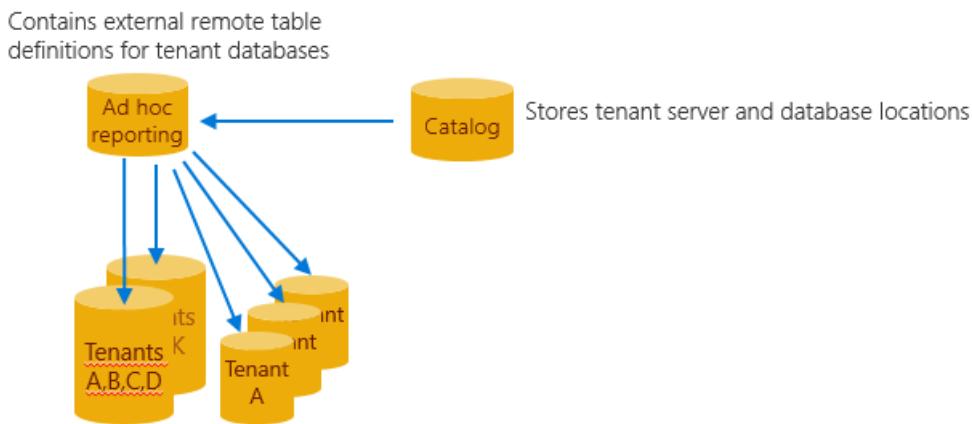
In this tutorial you learn:

- How to deploy an ad hoc reporting database
- How to run distributed queries across all tenant databases

To complete this tutorial, make sure the following prerequisites are completed:

- The Wingtip Tickets SaaS Multi-tenant Database app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Multi-tenant Database application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- SQL Server Management Studio (SSMS) is installed. To download and install SSMS, see [Download SQL Server Management Studio \(SSMS\)](#).

## Ad hoc reporting pattern



SaaS applications can analyze the vast amount of tenant data that is stored centrally in the cloud. The analyses reveal insights into the operation and usage of your application. These insights can guide feature development, usability improvements, and other investments in your apps and services.

Accessing this data in a single multi-tenant database is easy, but not so easy when distributed at scale across potentially thousands of databases. One approach is to use [Elastic Query](#), which enables querying across a distributed set of databases with common schema. These databases can be distributed across different resource groups and subscriptions. Yet one common login must have access to extract data from all the databases. Elastic Query uses a single *head* database in which external tables are defined that mirror tables or views in the distributed (tenant) databases. Queries submitted to this head database are compiled to produce a distributed query plan, with portions of the query pushed down to the tenant databases as needed. Elastic Query uses the shard map in the catalog database to determine the location of all tenant databases. Setup and query are straightforward using standard [Transact-SQL](#), and support ad hoc querying from tools like Power BI and Excel.

By distributing queries across the tenant databases, Elastic Query provides immediate insight into live production data. However, as Elastic Query pulls data from potentially many databases, query latency can sometimes be higher than for equivalent queries submitted to a single multi-tenant database. Be sure to design queries to minimize the data that is returned. Elastic Query is often best suited for querying small amounts of real-time data, as opposed to building frequently used or complex analytics queries or reports. If queries do not perform well, look at the [execution plan](#) to see what part of the query has been pushed down to the remote database. And assess how much data is being returned. Queries that require complex analytical processing might be better served by saving the extracted tenant data into a database that is optimized for analytics queries. SQL Database and SQL Data Warehouse could host such the analytics database.

This pattern for analytics is explained in the [tenant analytics tutorial](#).

## Get the Wingtip Tickets SaaS Multi-tenant Database application source code and scripts

The Wingtip Tickets SaaS Multi-tenant Database scripts and application source code are available in the [WingtipTicketsSaaS-MultitenantDB](#) GitHub repo. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

## Create ticket sales data

To run queries against a more interesting data set, create ticket sales data by running the ticket-generator.

1. In the *PowerShell ISE*, open the ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\*Demo-AdhocReporting.ps1* script and set the following values:
  - **\$DemoScenario = 1, Purchase tickets for events at all venues.**
2. Press **F5** to run the script and generate ticket sales. While the script is running, continue the steps in this tutorial.  
The ticket data is queried in the *Run ad hoc distributed queries* section, so wait for the ticket generator to complete.

## Explore the tenant tables

In the Wingtip Tickets SaaS Multi-tenant Database application, tenants are stored in a hybrid tenant management model - where tenant data is either stored in a multi-tenant database or a single tenant database and can be moved between the two. When querying across all tenant databases, it's important that Elastic Query can treat the data as if it is part of a single logical database sharded by tenant.

To achieve this pattern, all tenant tables include a *VenuelId* column that identifies which tenant the data belongs to. The *VenuelId* is computed as a hash of the Venue name, but any approach could be used to introduce a unique value for this column. This approach is similar to the way the tenant key is computed for use in the catalog. Tables containing *VenuelId* are used by Elastic Query to parallelize queries and push them down to the appropriate remote tenant database. This dramatically reduces the amount of data that is returned and results in an increase in performance especially when there are multiple tenants whose data is stored in single tenant databases.

## Deploy the database used for ad hoc distributed queries

This exercise deploys the *adhocreporting* database. This is the head database that contains the schema used for querying across all tenant databases. The database is deployed to the existing catalog server, which is the server used for all management-related databases in the sample app.

1. Open ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\*Demo-AdhocReporting.ps1* in the *PowerShell ISE* and set the following values:
  - **\$DemoScenario = 2, Deploy Ad hoc analytics database.**

2. Press **F5** to run the script and create the *adhocreporting* database.

In the next section, you add schema to the database so it can be used to run distributed queries.

## Configure the 'head' database for running distributed queries

This exercise adds schema (the external data source and external table definitions) to the ad hoc reporting database that enables querying across all tenant databases.

1. Open SQL Server Management Studio, and connect to the Adhoc reporting database you created in the previous step. The name of the database is *adhocreporting*.
2. Open ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\ *Initialize-AdhocReportingDB.sql* in SSMS.
3. Review the SQL script and note the following:

Elastic Query uses a database-scoped credential to access each of the tenant databases. This credential needs to be available in all the databases and should normally be granted the minimum rights required to enable these ad hoc queries.

```
-- Create login credential, used to access the catalog and remote databases
IF NOT EXISTS (SELECT * FROM sys.database_scoped_credentials WHERE name = 'AdhocQueryDBCred')
 CREATE DATABASE SCOPED CREDENTIAL [AdhocQueryDBCred] WITH IDENTITY = N'developer', SECRET = N'P@ssword1';
GO
```

By using the catalog database as the external data source, queries are distributed to all databases registered in the catalog when the query is run. Because server names are different for each deployment, this initialization script gets the location of the catalog database by retrieving the current server (@@servername) where the script is executed.

```
-- Add catalog database as external data source using credential created above
IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'WtpTenantDBs')
BEGIN
 DECLARE @catalogServerName nvarchar(128) = (SELECT @@servername + '.database.windows.net');
 DECLARE @createExternalSource nvarchar(500) =
 N'CREATE EXTERNAL DATA SOURCE [WtpTenantDBs]
 WITH
 (
 TYPE = SHARD_MAP_MANAGER,
 LOCATION = ''' + @catalogServerName + ''',
 DATABASE_NAME = ''tenantcatalog'',
 SHARD_MAP_NAME = ''tenantcatalog'',
 CREDENTIAL = [AdhocQueryDBCred]
);'
 EXEC(@createExternalSource)
END
```

The external tables that reference tenant tables are defined with **DISTRIBUTION = SHARDED(VenueId)**.

This routes a query for a particular *VenueId* to the appropriate database and improves performance for many scenarios as shown in the next section.

```
CREATE EXTERNAL TABLE [dbo].[VenueEvents]
(
 [VenueId] INT NOT NULL,
 [EventId] INT NOT NULL,
 [EventName] NVARCHAR (50) NOT NULL,
 [Subtitle] NVARCHAR (50) NULL,
 [Date] DATETIME NOT NULL
)
WITH
(
 DATA_SOURCE = [WtpTenantDBs],
 DISTRIBUTION = SHARDED(VenueId)
);
```

The local table *VenueTypes* that is created and populated. This reference data table is common in all tenant

databases, so it can be represented here as a local table and populated with the common data. For some queries, this may reduce the amount of data moved between the tenant databases and the *adhocreporting* database.

```
CREATE TABLE [dbo].[VenueTypes]
(
 [VenueType] CHAR(30) NOT NULL,
 [VenueTypeName] NCHAR(30) NOT NULL,
 [EventTypeName] NVARCHAR(30) NOT NULL,
 [EventTypeShortName] NVARCHAR(20) NOT NULL,
 [EventTypeShortNamePlural] NVARCHAR(20) NOT NULL,
 [Language] CHAR(8) NOT NULL,
 PRIMARY KEY CLUSTERED ([VenueType] ASC)
)
```

If you include reference tables in this manner, be sure to update the table schema and data whenever you update the tenant databases.

4. Press **F5** to run the script and initialize the *adhocreporting* database.

Now you can run distributed queries, and gather insights across all tenants!

## Run ad hoc distributed queries

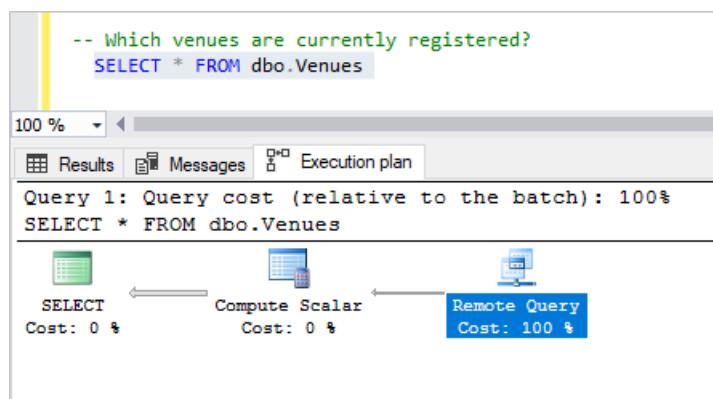
Now that the *adhocreporting* database is set up, go ahead and run some distributed queries. Include the execution plan for a better understanding of where the query processing is happening.

When inspecting the execution plan, hover over the plan icons for details.

1. In SSMS, open ...\\Learning Modules\\Operational Analytics\\Adhoc Reporting\\*Demo-AdhocReportingQueries.sql*.
2. Ensure you are connected to the **adhocreporting** database.
3. Select the **Query** menu and click **Include Actual Execution Plan**
4. Highlight the *Which venues are currently registered?* query, and press **F5**.

The query returns the entire venue list, illustrating how quick and easy it is to query across all tenants and return data from each tenant.

Inspect the plan and see that the entire cost is the remote query because we're simply going to each tenant database and selecting the venue information.



5. Select the next query, and press **F5**.

This query joins data from the tenant databases and the local *VenueTypes* table (local, as it's a table in the *adhocreporting* database).

Inspect the plan and see that the majority of cost is the remote query because we query each tenant's venue info (dbo.Venues), and then do a quick local join with the local *VenueTypes* table to display the friendly

name.

```
-- Which venues are currently registered?
SELECT * FROM dbo.Venues

GO

-- And what is their venue type?
SELECT VenueName,
 VenueTypeName,
 EventType
FROM dbo.Venues
 INNER JOIN dbo.VenueTypes ON Venues.VenueType = VenueTypes.VenueType
```

100 % ▾ Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

```
SELECT VenueName, VenueTypeName, EventType FROM dbo.Venues INNER JOIN c...
Clustered Index Scan (Clustered) [VenueTypes].[PK_VenueTyp_265E44F... Cost: 1 %
Hash Match (Inner Join) Cost: 7 %
Compute Scalar Cost: 0 %
Remote Query Cost: 92 %
```

- Now select the *On which day were the most tickets sold?* query, and press **F5**.

This query does a bit more complex joining and aggregation. What's important to note is that most of the processing is done remotely, and once again, we bring back only the rows we need, returning just a single row for each venue's aggregate ticket sale count per day.

```
-- On which day were the most tickets sold?
SELECT CAST(PurchaseDate AS DATE) AS TicketPurchaseDate,
 Count(TicketId) AS TicketCount
FROM VenueTicketPurchases
 INNER JOIN VenueTickets ON (VenueTickets.TicketPurchaseId = VenueTicketPurchases.TicketPurchaseId AND VenueTickets.VenueId = VenueTicketPurchases.VenueId)
GROUP BY (CAST(PurchaseDate AS DATE))
ORDER BY TicketCount DESC, TicketPurchaseDate ASC
```

100 % ▾ Results Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

```
SELECT CAST(PurchaseDate AS DATE) AS TicketPurchaseDate, Count(TicketId) AS TicketCount FROM VenueTicketPurchases
Sort Cost: 27 %
Compute Scalar Cost: 0 %
Stream Aggregate (Aggregate) Cost: 0 %
Sort Cost: 27 %
Compute Scalar Cost: 0 %
Remote Query Cost: 47 %
```

## Next steps

In this tutorial you learned how to:

- Run distributed queries across all tenant databases
- Deploy an ad hoc reporting database and add schema to it to run distributed queries.

Now try the [Tenant Analytics tutorial](#) to explore extracting data to a separate analytics database for more complex analytics processing.

## Additional resources

- [Elastic Query](#)

# Manage schema in a SaaS application that uses sharded multi-tenant SQL databases

11/7/2019 • 7 minutes to read • [Edit Online](#)

This tutorial examines the challenges in maintaining a fleet of databases in a Software as a Service (SaaS) application. Solutions are demonstrated for fanning out schema changes across the fleet of databases.

Like any application, the Wingtip Tickets SaaS app will evolve over time, and will require changes to the database. Changes may impact schema or reference data, or apply database maintenance tasks. With a SaaS application using a database per tenant pattern, changes must be coordinated across a potentially massive fleet of tenant databases. In addition, you must incorporate these changes into the database provisioning process to ensure they are included in new databases as they are created.

## Two scenarios

This tutorial explores the following two scenarios:

- Deploy reference data updates for all tenants.
- Rebuild an index on the table that contains the reference data.

The [Elastic Jobs](#) feature of Azure SQL Database is used to execute these operations across tenant databases. The jobs also operate on the 'template' tenant database. In the Wingtip Tickets sample app, this template database is copied to provision a new tenant database.

In this tutorial you learn how to:

- Create a job agent.
- Execute a T-SQL query on multiple tenant databases.
- Update reference data in all tenant databases.
- Create an index on a table in all tenant databases.

## Prerequisites

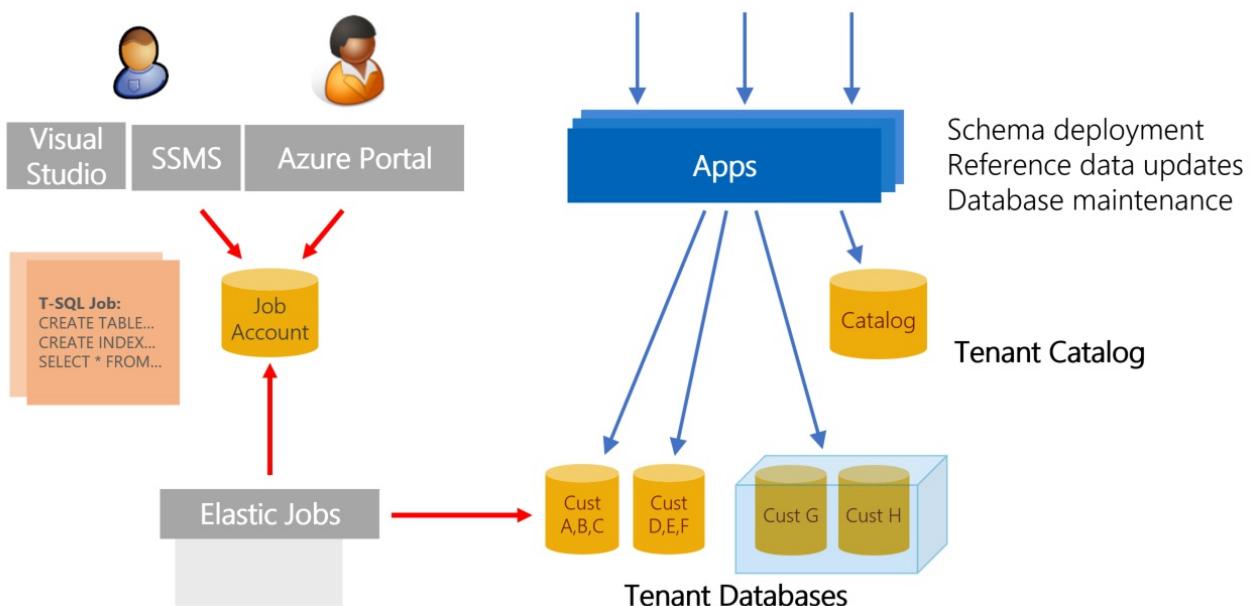
- The Wingtip Tickets multi-tenant database app must already be deployed:
  - For instructions, see the first tutorial, which introduces the Wingtip Tickets SaaS multi-tenant database app:  
[Deploy and explore a sharded multi-tenant application that uses Azure SQL Database](#).
    - The deploy process runs for less than five minutes.
  - You must have the *sharded multi-tenant* version of Wingtip installed. The versions for *Standalone* and *Database per tenant* do not support this tutorial.
- The latest version of SQL Server Management Studio (SSMS) must be installed. [Download and Install SSMS](#).
- Azure PowerShell must be installed. For details, see [Getting started with Azure PowerShell](#).

#### NOTE

This tutorial uses features of the Azure SQL Database service that are in a limited preview ([Elastic Database jobs](#)). If you wish to do this tutorial, provide your subscription ID to [SaaSFeedback@microsoft.com](mailto:SaaSFeedback@microsoft.com) with subject=Elastic Jobs Preview. After you receive confirmation that your subscription has been enabled, [download and install the latest pre-release jobs cmdlets](#). This preview is limited, so contact [SaaSFeedback@microsoft.com](mailto:SaaSFeedback@microsoft.com) for related questions or support.

## Introduction to SaaS schema management patterns

The sharded multi-tenant database model used in this sample enables a tenants database to contain one or more tenants. This sample explores the potential to use a mix of a many-tenant and one-tenant databases, enabling a *hybrid* tenant management model. Managing changes to these databases can be complicated. [Elastic Jobs](#) facilitates administration and management of large numbers of database. Jobs enable you to securely and reliably run Transact-SQL scripts as tasks, against a group of tenant databases. The tasks are independent of user interaction or input. This method can be used to deploy changes to schema or to common reference data, across all tenants in an application. Elastic Jobs can also be used to maintain a golden template copy of the database. The template is used to create new tenants, always ensuring the latest schema and reference data are in use.



## Elastic Jobs limited preview

There is a new version of Elastic Jobs that is now an integrated feature of Azure SQL Database. This new version of Elastic Jobs is currently in limited preview. The limited preview currently supports using PowerShell to create a job agent, and T-SQL to create and manage jobs.

#### NOTE

This tutorial uses features of the SQL Database service that are in a limited preview (Elastic Database jobs). If you wish to do this tutorial, provide your subscription ID to [SaaSFeedback@microsoft.com](mailto:SaaSFeedback@microsoft.com) with subject=Elastic Jobs Preview. After you receive confirmation that your subscription has been enabled, download and install the latest pre-release jobs cmdlets. This preview is limited, so contact [SaaSFeedback@microsoft.com](mailto:SaaSFeedback@microsoft.com) for related questions or support.

## Get the Wingtip Tickets SaaS Multi-tenant Database application source code and scripts

The Wingtip Tickets SaaS Multi-tenant Database scripts and application source code are available in the [WingtipTicketsSaaS-MultitenantDB](#) repository on GitHub. See the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.

## Create a job agent database and new job agent

This tutorial requires that you use PowerShell to create the job agent database and job agent. Like the MSDB database used by SQL Agent, a job agent uses an Azure SQL database to store job definitions, job status, and history. After the job agent is created, you can create and monitor jobs immediately.

1. In **PowerShell ISE**, open ...\\Learning Modules\\Schema Management\\Demo-SchemaManagement.ps1.
2. Press **F5** to run the script.

The *Demo-SchemaManagement.ps1* script calls the *Deploy-SchemaManagement.ps1* script to create a database named *jobagent* on the catalog server. The script then creates the job agent, passing the *jobagent* database as a parameter.

## Create a job to deploy new reference data to all tenants

### Prepare

Each tenant's database includes a set of venue types in the **VenueTypes** table. Each venue type defines the kind of events that can be hosted at a venue. These venue types correspond to the background images you see in the tenant events app. In this exercise, you deploy an update to all databases to add two additional venue types: *Motorcycle Racing* and *Swimming Club*.

First, review the venue types included in each tenant database. Connect to one of the tenant databases in SQL Server Management Studio (SSMS) and inspect the *VenueTypes* table. You can also query this table in the Query editor in the Azure portal, accessed from the database page.

1. Open SSMS and connect to the tenant server: *tenants1-dpt-<user>.database.windows.net*
2. To confirm that *Motorcycle Racing* and *Swimming Club* **are not** currently included, browse to the *contosoconcerthall* database on the *tenants1-dpt-<user>* server and query the *VenueTypes* table.

### Steps

Now you create a job to update the **VenueTypes** table in each tenants database, by adding the two new venue types.

To create a new job, you use the set of jobs system stored procedures that were created in the *jobagent* database. The stored procedures were created when the job agent was created.

1. In SSMS, connect to the tenant server: *tenants1-mt-<user>.database.windows.net*
2. Browse to the *tenants1* database.
3. Query the *VenueTypes* table to confirm that *Motorcycle Racing* and *Swimming Club* are not yet in the results list.
4. Connect to the catalog server, which is *catalog-mt-<user>.database.windows.net*.
5. Connect to the *jobagent* database in the catalog server.
6. In SSMS, open the file ...\\Learning Modules\\Schema Management\\DeployReferenceData.sql.
7. Modify the statement: set @User = <user> and substitute the User value used when you deployed the Wingtip Tickets SaaS Multi-tenant Database application.
8. Press **F5** to run the script.

### Observe

Observe the following items in the *DeployReferenceData.sql* script:

- **sp\_add\_target\_group** creates the target group name *DemoServerGroup*, and adds target members to the group.
- **sp\_add\_target\_group\_member** adds the following items:
  - A *server* target member type.
    - This is the *tenants1-mt-<user>* server that contains the tenants databases.
    - Including the server includes the tenant databases that exist at the time the job executes.
  - A *database* target member type for the template database (*basetenantdb*) that resides on *catalog-mt-<user>* server,
  - A *database* target member type to include the *adhocreporting* database that is used in a later tutorial.
- **sp\_add\_job** creates a job called *Reference Data Deployment*.
- **sp\_add\_jobstep** creates the job step containing T-SQL command text to update the reference table, *VenueTypes*.
- The remaining views in the script display the existence of the objects and monitor job execution. Use these queries to review the status value in the **lifecycle** column to determine when the job has finished. The job updates the tenants database, and updates the two additional databases that contain the reference table.

In SSMS, browse to the tenant database on the *tenants1-mt-<user>* server. Query the *VenueTypes* table to confirm that *Motorcycle Racing* and *Swimming Club* are now added to the table. The total count of venue types should have increased by two.

## Create a job to manage the reference table index

This exercise creates a job to rebuild the index on the reference table primary key on all the tenant databases. An index rebuild is a typical database management operation that an administrator might run after loading a large amount of data load, to improve performance.

1. In SSMS, connect to *jobagent* database in *catalog-mt-<User>.database.windows.net* server.
2. In SSMS, open ...\\Learning Modules\\Schema Management\\*OnlineReindex.sql*.
3. Press **F5** to run the script.

### Observe

Observe the following items in the *OnlineReindex.sql* script:

- **sp\_add\_job** creates a new job called *Online Reindex PK\_VenueTyp\_265E44FD7FD4C885*.
- **sp\_add\_jobstep** creates the job step containing T-SQL command text to update the index.
- The remaining views in the script monitor job execution. Use these queries to review the status value in the **lifecycle** column to determine when the job has successfully finished on all target group members.

## Additional resources

- [Managing scaled-out cloud databases](#)

## Next steps

In this tutorial you learned how to:

- Create a job agent to run T-SQL jobs across multiple databases
- Update reference data in all tenant databases

- Create an index on a table in all tenant databases

Next, try the [Ad hoc reporting tutorial](#) to explore running distributed queries across tenant databases.

# Cross-tenant analytics using extracted data - multi-tenant app

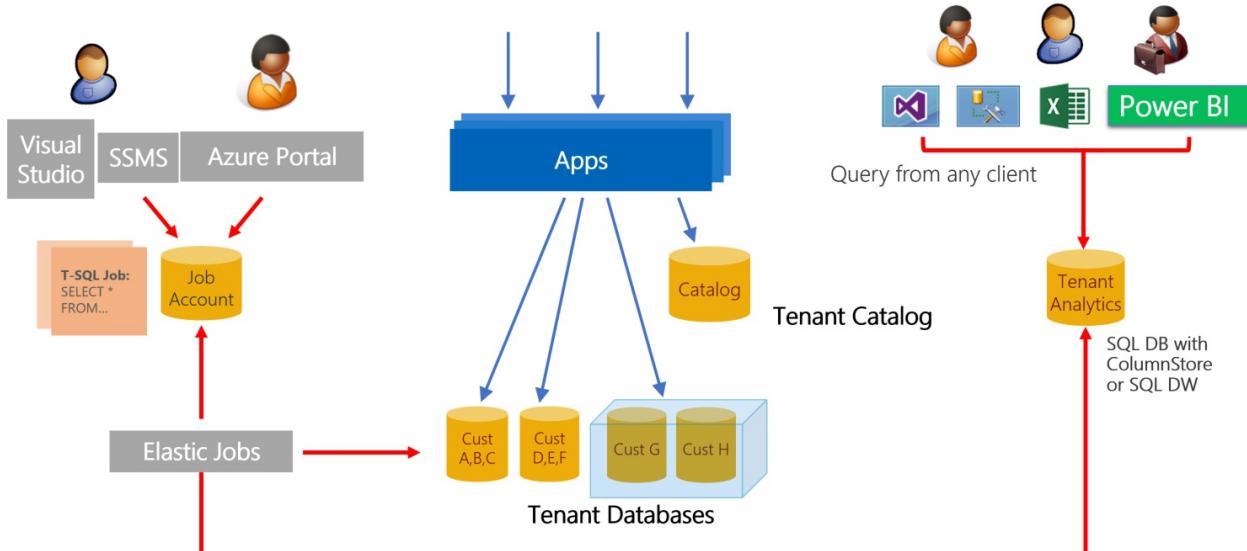
11/7/2019 • 13 minutes to read • [Edit Online](#)

In this tutorial, you walk through a complete analytics scenario for a multitenant implementation. The scenario demonstrates how analytics can enable businesses to make smart decisions. Using data extracted from sharded database, you use analytics to gain insights into tenant behavior, including their use of the sample Wingtip Tickets SaaS application. This scenario involves three steps:

1. **Extract data** from each tenant database into an analytics store.
2. **Optimize the extracted data** for analytics processing.
3. Use **Business Intelligence** tools to draw out useful insights, which can guide decision making.

In this tutorial you learn how to:

- Create the tenant analytics store to extract the data into.
- Use elastic jobs to extract data from each tenant database into the analytics store.
- Optimize the extracted data (reorganize into a star-schema).
- Query the analytics database.
- Use Power BI for data visualization to highlight trends in tenant data and make recommendation for improvements.



## Offline tenant analytics pattern

SaaS applications you develop have access to a vast amount of tenant data stored in the cloud. The data provides a rich source of insights about the operation and usage of your application, and about the behavior of the tenants. These insights can guide feature development, usability improvements, and other investments in the app and platform.

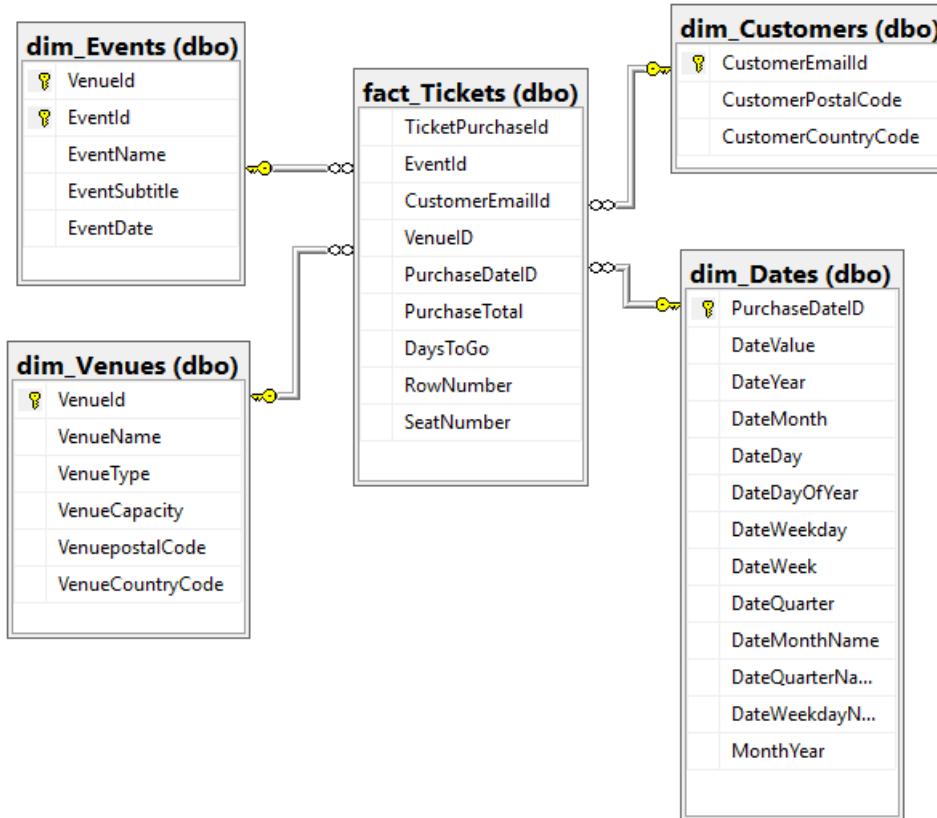
Accessing the data for all tenants is simple when all the data is in just one multi-tenant database. But the access is more complex when distributed at scale across thousands of databases. One way to tame the complexity is to extract the data to an analytics database or a data warehouse. You then query the data warehouse to gather insights from the tickets data of all tenants.

This tutorial presents a complete analytics scenario for this sample SaaS application. First, elastic jobs are used to schedule the extraction of data from each tenant database. The data is sent to an analytics store. The analytics store could either be an SQL Database or a SQL Data Warehouse. For large-scale data extraction, [Azure Data Factory](#) is commended.

Next, the aggregated data is shredded into a set of [star-schema](#) tables. The tables consist of a central fact table plus related dimension tables:

- The central fact table in the star-schema contains ticket data.
- The dimension tables contain data about venues, events, customers, and purchase dates.

Together the central and dimension tables enable efficient analytical processing. The star-schema used in this tutorial is displayed in the following image:



Finally, the star-schema tables are queried. The query results are displayed visually to highlight insights into tenant behavior and their use of the application. With this star-schema, you can run queries that help discover items like the following:

- Who is buying tickets and from which venue.
- Hidden patterns and trends in the following areas:
  - The sales of tickets.
  - The relative popularity of each venue.

Understanding how consistently each tenant is using the service provides an opportunity to create service plans to cater to their needs. This tutorial provides basic examples of insights that can be gleaned from tenant data.

## Setup

### Prerequisites

To complete this tutorial, make sure the following prerequisites are met:

- The Wingtip Tickets SaaS Multi-tenant Database application is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Multi-tenant Database application](#)

- The Wingtip SaaS scripts and application [source code](#) are downloaded from GitHub. Be sure to *unblock the zip file* before extracting its contents. Check out the [general guidance](#) for steps to download and unblock the Wingtip Tickets SaaS scripts.
- Power BI Desktop is installed. [Download Power BI Desktop](#)
- The batch of additional tenants has been provisioned, see the [Provision tenants tutorial](#).
- A job agent and job agent database have been created. See the appropriate steps in the [Schema management tutorial](#).

### Create data for the demo

In this tutorial, analysis is performed on ticket sales data. In the current step, you generate ticket data for all the tenants. Later this data is extracted for analysis. *Ensure you have provisioned the batch of tenants as described earlier, so that you have a meaningful amount of data.* A sufficiently large amount of data can expose a range of different ticket purchasing patterns.

1. In **PowerShell ISE**, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\Demo-TenantAnalytics.ps1, and set the following value:
  - **\$DemoScenario = 1** Purchase tickets for events at all venues
2. Press **F5** to run the script and create ticket purchasing history for every event in each venue. The script runs for several minutes to generate tens of thousands of tickets.

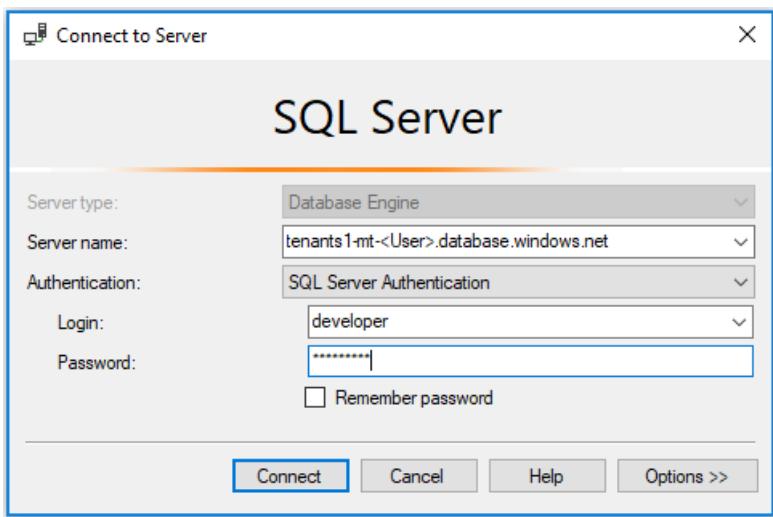
### Deploy the analytics store

Often there are numerous transactional sharded databases that together hold all tenant data. You must aggregate the tenant data from the sharded database into one analytics store. The aggregation enables efficient query of the data. In this tutorial, an Azure SQL Database database is used to store the aggregated data.

In the following steps, you deploy the analytics store, which is called **tenantanalytics**. You also deploy predefined tables that are populated later in the tutorial:

1. In PowerShell ISE, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\Demo-TenantAnalytics.ps1
2. Set the \$DemoScenario variable in the script to match your choice of analytics store. For learning purposes, SQL database without columnstore is recommended.
  - To use SQL database without columnstore, set **\$DemoScenario = 2**
  - To use SQL database with columnstore, set **\$DemoScenario = 3**
3. Press **F5** to run the demo script (that calls the Deploy-TenantAnalytics<XX>.ps1 script) which creates the tenant analytics store.

Now that you have deployed the application and filled it with interesting tenant data, use [SQL Server Management Studio \(SSMS\)](#) to connect **tenants1-mt-<User>** and **catalog-mt-<User>** servers using Login = *developer*, Password = *P@ssword1*.

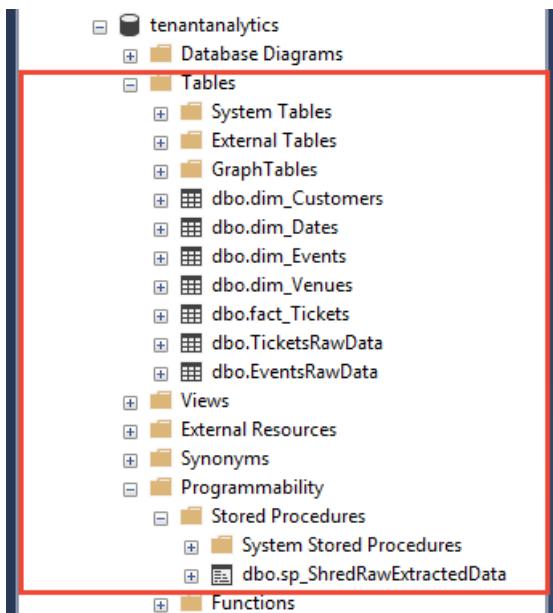


In the Object Explorer, perform the following steps:

1. Expand the *tenants1-mt-<User>* server.
2. Expand the Databases node, and see *tenants1* database containing multiple tenants.
3. Expand the *catalog-mt-<User>* server.
4. Verify that you see the analytics store and the jobaccount database.

See the following database items in the SSMS Object Explorer by expanding the analytics store node:

- Tables **TicketsRawData** and **EventsRawData** hold raw extracted data from the tenant databases.
- The star-schema tables are **fact\_Tickets**, **dim\_Customers**, **dim\_Venues**, **dim\_Events**, and **dim\_Dates**.
- The **sp\_ShredRawExtractedData** stored procedure is used to populate the star-schema tables from the raw data tables.



## Data extraction

### Create target groups

Before proceeding, ensure you have deployed the job account and jobaccount database. In the next set of steps, Elastic Jobs is used to extract data from the sharded tenants database, and to store the data in the analytics store. Then the second job shreds the data and stores it into tables in the star-schema. These two jobs run against two different target groups, namely **TenantGroup** and **AnalyticsGroup**. The extract job runs against the TenantGroup, which contains all the tenant databases. The shredding job runs against the AnalyticsGroup, which contains just the analytics store. Create the target groups by using the following steps:

1. In SSMS, connect to the **jobaccount** database in catalog-mt-<User>.
2. In SSMS, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\TargetGroups.sql
3. Modify the @User variable at the top of the script, replacing <User> with the user value used when you deployed the Wingtip Tickets SaaS Multi-tenant Database application.
4. Press **F5** to run the script that creates the two target groups.

### Extract raw data from all tenants

Transactions might occur more frequently for *ticket and customer* data than for *event and venue* data. Therefore, consider extracting ticket and customer data separately and more frequently than you extract event and venue data. In this section, you define and schedule two separate jobs:

- Extract ticket and customer data.
- Extract event and venue data.

Each job extracts its data, and posts it into the analytics store. There a separate job shreds the extracted data into the analytics star-schema.

1. In SSMS, connect to the **jobaccount** database in catalog-mt-<User> server.
2. In SSMS, open ...\\Learning Modules\\Operational Analytics\\Tenant Analytics\\ExtractTickets.sql.
3. Modify @User at the top of the script, and replace <User> with the user name used when you deployed the Wingtip Tickets SaaS Multi-tenant Database application.
4. Press **F5** to run the script that creates and runs the job that extracts tickets and customers data from each tenant database. The job saves the data into the analytics store.
5. Query the TicketsRawData table in the tenantanalytics database, to ensure that the table is populated with tickets information from all tenants.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including databases like tenants1-mt-aj3.database.windows.net and catalog-mt-aj3.database.windows.net, and tables such as dbo.TicketsRawData. The central pane displays a T-SQL script named ExtractTickets.sql, which creates and runs a job to extract ticket data. The Results pane below shows the output of the query, which is a table of job executions. The table has columns: job\_execution\_id, job\_name, job\_id, job\_ver..., step\_id, is\_active, lifecycle, and create\_time. Three rows are shown, all with a lifecycle of 'Succeeded' and a create\_time of '2017-11-15'. The status bar at the bottom indicates the query was executed successfully.

	job_execution_id	job_name	job_id	job_ver...	step_id	is_active	lifecycle	create_time
1	BD2AE4A8-9D2...	ExtractTickets	0FEB5383-F...	1	NULL	0	Succeeded	2017-11-15
2	BD2AE4A8-9D2...	ExtractTickets	0FEB5383-F...	1	1	0	Succeeded	2017-11-15
3	BD2AE4A8-9D2...	ExtractTickets	0FEB5383-F...	1	1	0	Succeeded	2017-11-15

Repeat the preceding steps, except this time replace \ExtractTickets.sql with \ExtractVenuesEvents.sql in step 2.

Successfully running the job populates the EventsRawData table in the analytics store with new events and venues information from all tenants.

## Data reorganization

### Shred extracted data to populate star-schema tables

The next step is to shred the extracted raw data into a set of tables that are optimized for analytics queries. A star-schema is used. A central fact table holds individual ticket sales records. Dimension tables are populated with data about venues, events, customers, and purchase dates.

In this section of the tutorial, you define and run a job that merges the extracted raw data with the data in the star-schema tables. After the merge job is finished, the raw data is deleted, leaving the tables ready to be populated by the next tenant data extract job.

1. In SSMS, connect to the **jobaccount** database in catalog-mt-<User>.
2. In SSMS, open ... \Learning Modules\Operational Analytics\Tenant Analytics\ShredRawExtractedData.sql.
3. Press **F5** to run the script to define a job that calls the sp\_ShredRawExtractedData stored procedure in the analytics store.
4. Allow enough time for the job to run successfully.
  - Check the **Lifecycle** column of jobs.jobs\_execution table for the status of job. Ensure that the job **Succeeded** before proceeding. A successful run displays data similar to the following chart:

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the catalog-mt-aj3 database and its tables: dbo.dim\_Customers, dbo.dim\_Dates, dbo.dim\_Events, dbo.dim\_Venues, dbo.EventsRawData, dbo.fact\_Tickets, and dbo.TicketsRawData. The Results tab on the right displays the execution status of a job named 'ShredRawExtractedData'. The results table shows three rows of data with columns: job\_execution\_id, job\_name, job\_id, job\_v..., step\_id, is\_active, lifecycle, and create. All rows show a lifecycle of 'Succeeded' and a creation year of 2017.

job_execution_id	job_name	job_id	job_v...	step_id	is_active	lifecycle	create
1	3FC8B2B8-EBDB-4E7...	ShredRawExt...	4E3FB1D2-402...	1	1	0	Succeeded 2017
2	3FC8B2B8-EBDB-4E7...	ShredRawExt...	4E3FB1D2-402...	1	NULL	0	Succeeded 2017
3	3FC8B2B8-EBDB-4E7...	ShredRawExt...	4E3FB1D2-402...	1	1	0	Succeeded 2017

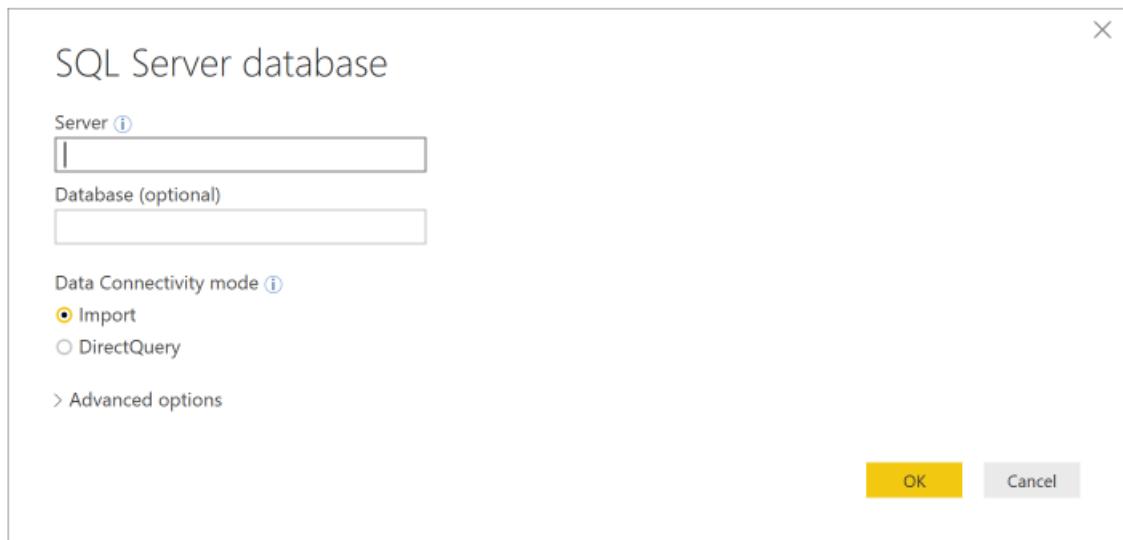
## Data exploration

### Visualize tenant data

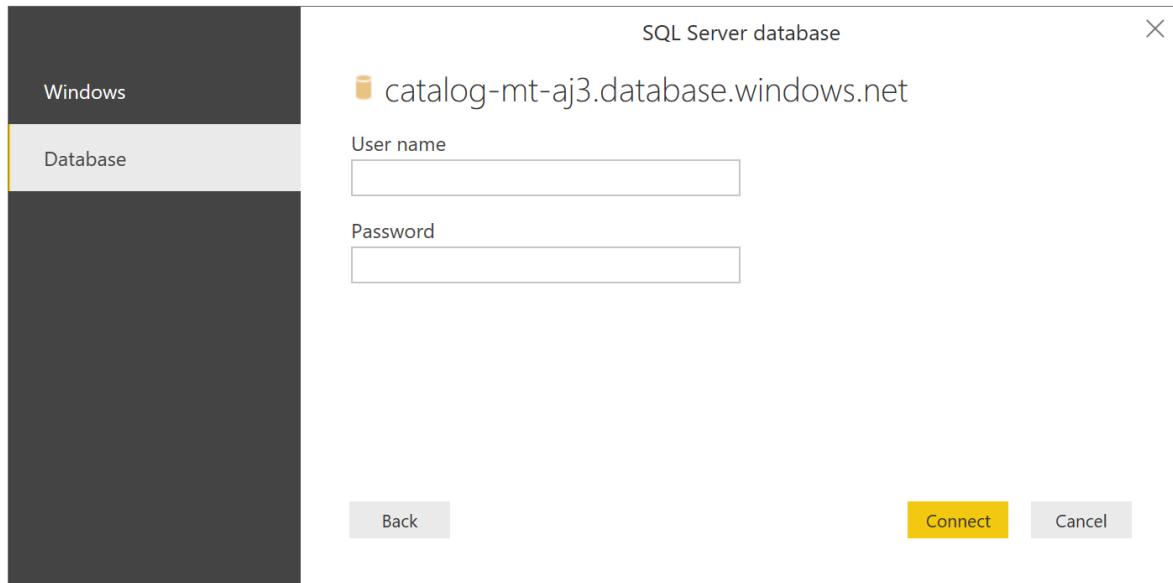
The data in the star-schema table provides all the ticket sales data needed for your analysis. To make it easier to see trends in large data sets, you need to visualize it graphically. In this section, you learn how to use **Power BI** to manipulate and visualize the tenant data you have extracted and organized.

Use the following steps to connect to Power BI, and to import the views you created earlier:

1. Launch Power BI desktop.
2. From the Home ribbon, select **Get Data**, and select **More...** from the menu.
3. In the **Get Data** window, select Azure SQL Database.
4. In the database login window, enter your server name (catalog-mt-<User>.database.windows.net). Select **Import** for **Data Connectivity Mode**, and then click OK.



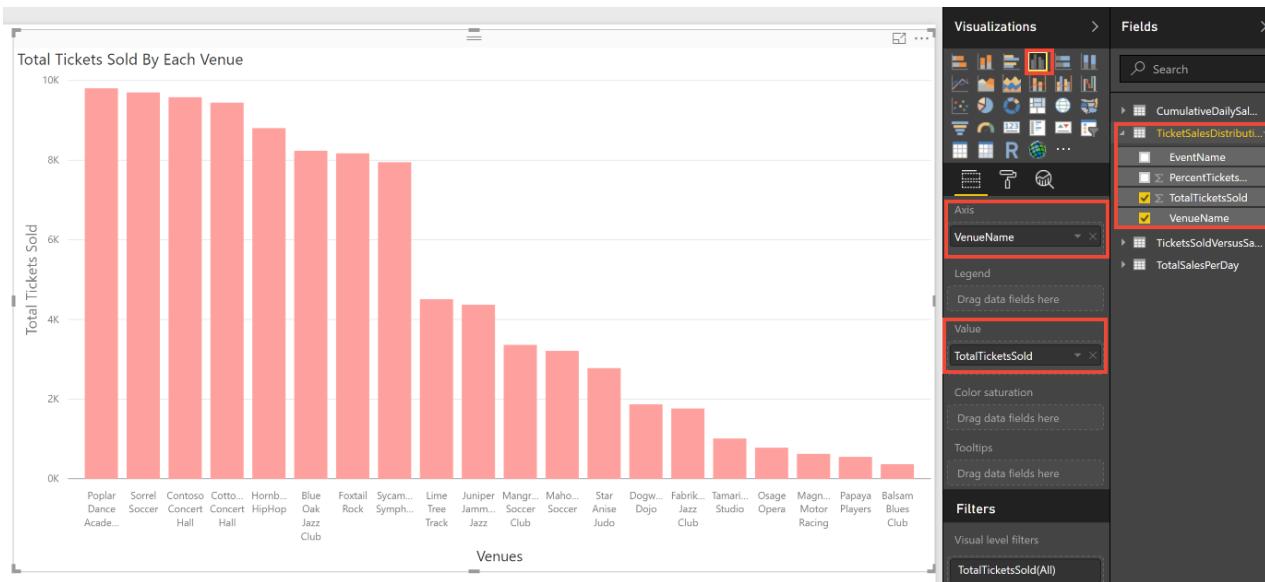
5. Select **Database** in the left pane, then enter user name = *developer*, and enter password = *P@ssword1*. Click **Connect**.



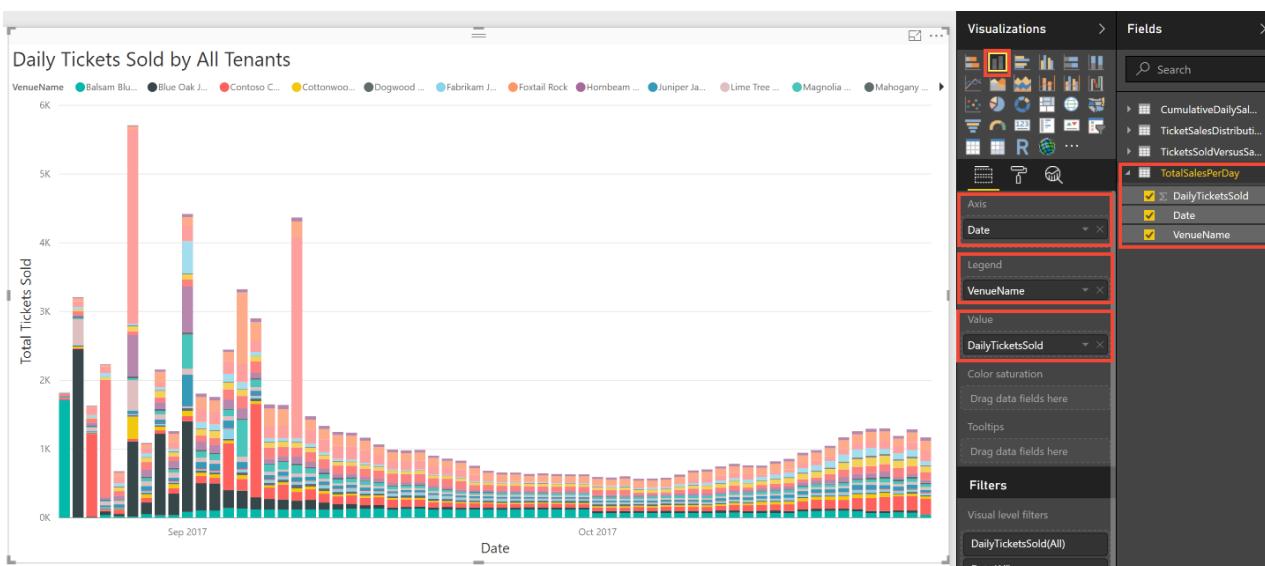
6. In the **Navigator** pane, under the analytics database, select the star-schema tables: fact\_Tickets, dim\_Events, dim\_Venues, dim\_Customers and dim\_Dates. Then select **Load**.

Congratulations! You have successfully loaded the data into Power BI. Now you can start exploring interesting visualizations to help gain insights into your tenants. Next you walk through how analytics can enable you to provide data-driven recommendations to the Wingtip Tickets business team. The recommendations can help to optimize the business model and customer experience.

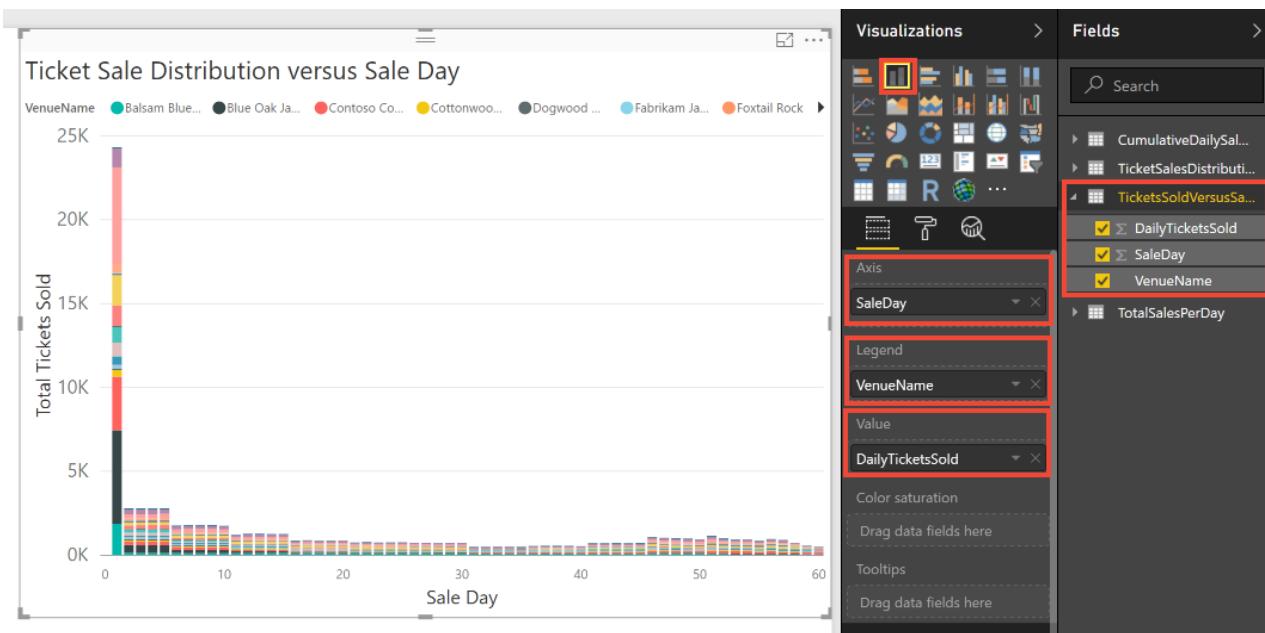
You start by analyzing ticket sales data to see the variation in usage across the venues. Select the following options in Power BI to plot a bar chart of the total number of tickets sold by each venue. Due to random variation in the ticket generator, your results may be different.



You can further analyze the data to see how ticket sales vary over time. Select the following options in Power BI to plot the total number of tickets sold each day for a period of 60 days.

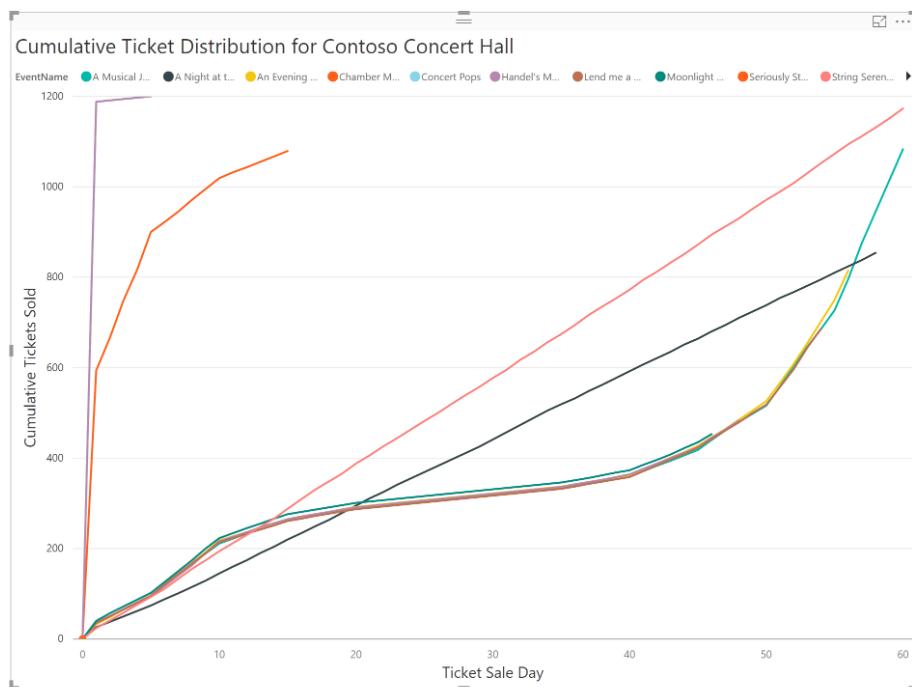


Next you want to further investigate the significance of these peak sale days. When do these peaks occur after tickets go on sale? To plot tickets sold per day, select the following options in Power BI.



The preceding plot shows that some venues sell a lot of tickets on the first day of sale. As soon as tickets go on sale at these venues, there seems to be a mad rush. This burst of activity by a few venues might impact the service for other tenants.

You can drill into the data again to see if this mad rush is true for all events hosted by these venues. In previous plots, you observed that Contoso Concert Hall sells a lot of tickets, and that Contoso also has a spike in ticket sales on certain days. Play around with Power BI options to plot cumulative ticket sales for Contoso Concert Hall, focusing on sale trends for each of its events. Do all events follow the same sale pattern?



The preceding plot for Contoso Concert Hall shows that the mad rush does not happen for all events. Play around with the filter options to see sale trends for other venues.

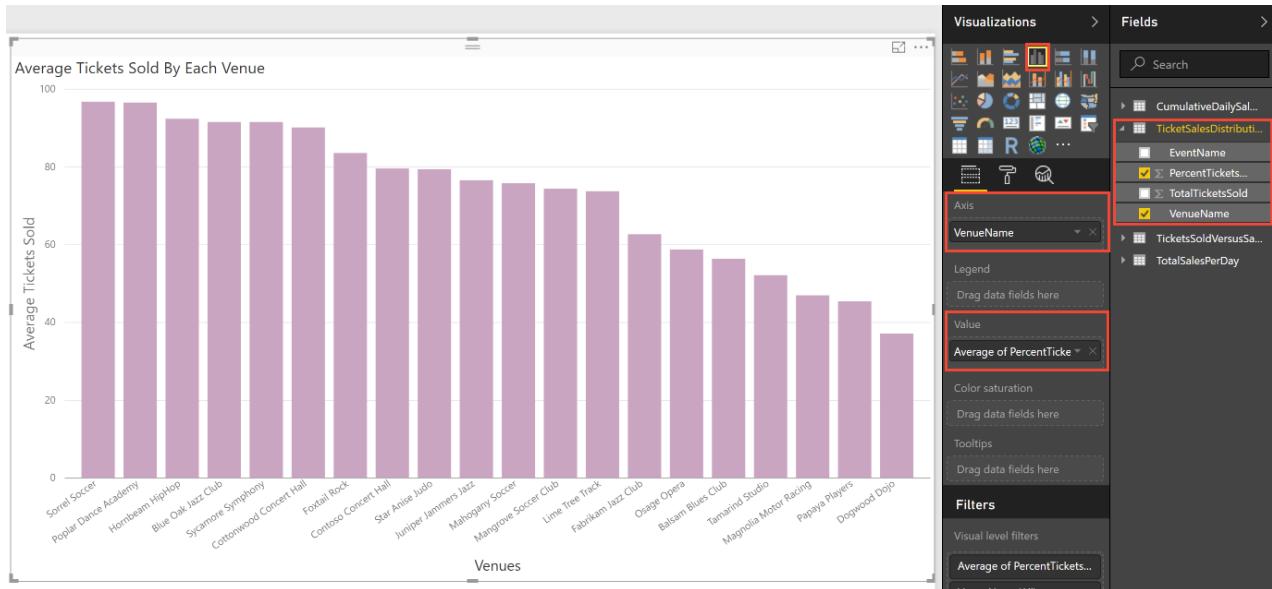
The insights into ticket selling patterns might lead Wingtip Tickets to optimize their business model. Instead of charging all tenants equally, perhaps Wingtip should introduce service tiers with different compute sizes. Larger venues that need to sell more tickets per day could be offered a higher tier with a higher service level agreement (SLA). Those venues could have their databases placed in pool with higher per-database resource limits. Each service tier could have an hourly sales allocation, with additional fees charged for exceeding the allocation. Larger

venues that have periodic bursts of sales would benefit from the higher tiers, and Wingtip Tickets can monetize their service more efficiently.

Meanwhile, some Wingtip Tickets customers complain that they struggle to sell enough tickets to justify the service cost. Perhaps in these insights there is an opportunity to boost ticket sales for under performing venues. Higher sales would increase the perceived value of the service. Right click fact\_Tickets and select **New measure**. Enter the following expression for the new measure called **AverageTicketsSold**:

```
AverageTicketsSold = DIVIDE(DIVIDE(COUNTROWS(fact_Tickets),DISTINCT(dim_Venues[VenueCapacity]))*100,
COUNTROWS(dim_Events))
```

Select the following visualization options to plot the percentage tickets sold by each venue to determine their relative success.



The preceding plot shows that even though most venues sell more than 80% of their tickets, some are struggling to fill more than half the seats. Play around with the Values Well to select maximum or minimum percentage of tickets sold for each venue.

Earlier you deepened your analysis to discover that ticket sales tend to follow predictable patterns. This discovery might let Wingtip Tickets help underperforming venues boost ticket sales by recommending dynamic pricing. This discovery could reveal an opportunity to employ machine learning techniques to predict ticket sales for each event. Predictions could also be made for the impact on revenue of offering discounts on ticket sales. Power BI Embedded could be integrated into an event management application. The integration could help visualize predicted sales and the effect of different discounts. The application could help devise an optimum discount to be applied directly from the analytics display.

You have observed trends in tenant data from the Wingtip Tickets SaaS Multi-tenant Database application. You can contemplate other ways the app can inform business decisions for SaaS application vendors. Vendors can better cater to the needs of their tenants. Hopefully this tutorial has equipped you with tools necessary to perform analytics on tenant data to empower your businesses to make data-driven decisions.

## Next steps

In this tutorial, you learned how to:

- Deployed a tenant analytics database with pre-defined star schema tables
- Used elastic jobs to extract data from all the tenant database
- Merge the extracted data into tables in a star-schema designed for analytics

- Query an analytics database
- Use Power BI for data visualization to observe trends in tenant data

Congratulations!

## Additional resources

Additional [tutorials that build upon the Wingtip SaaS application](#).

- [Elastic Jobs](#).
- [Cross-tenant analytics using extracted data - single-tenant app](#)

# Request quota increases for Azure SQL Database

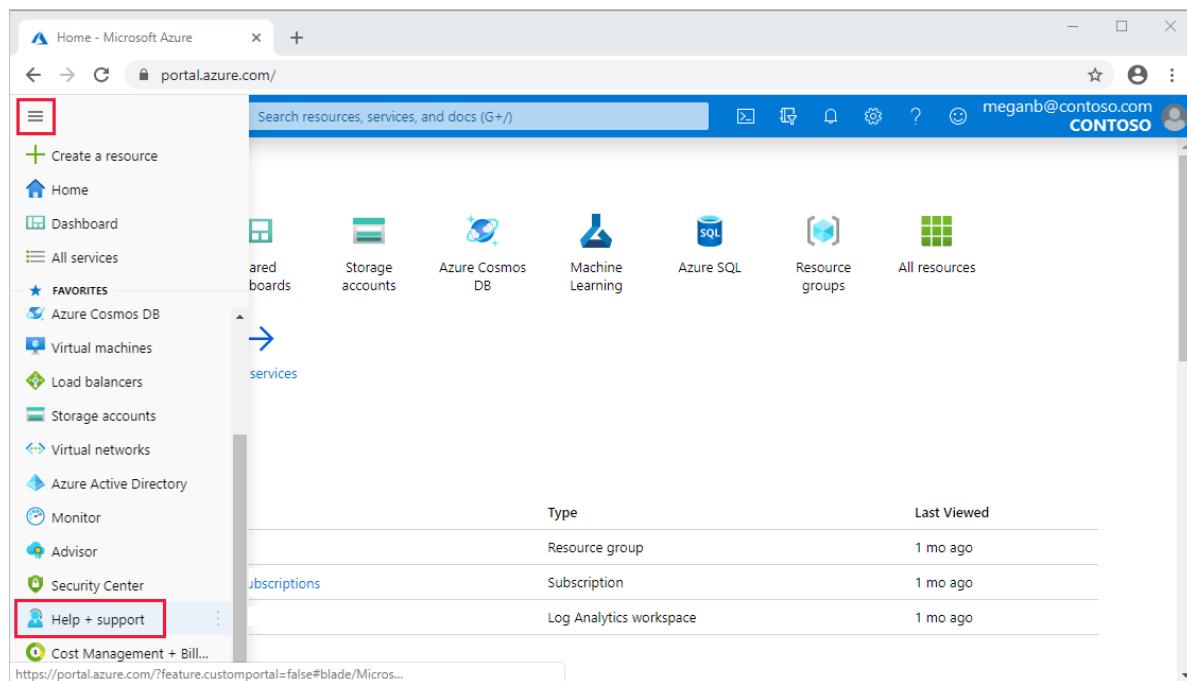
2/24/2020 • 2 minutes to read • [Edit Online](#)

This article explains how to request a quota increase for Azure SQL Database for single databases, servers, and managed instances. It also explains how to enable subscription access to a region.

## Create a new support request

Use the following steps to create a new support request from the Azure portal for SQL Database.

1. On the [Azure portal](#) menu, select **Help + support**.



The screenshot shows the Microsoft Azure portal homepage. The left sidebar has a red box around the 'Help + support' link under the 'Favorites' section. The main content area shows various service icons: Boards, Storage accounts, Azure Cosmos DB, Machine Learning, Azure SQL, Resource groups, and All resources. Below these is a 'services' section with a right-pointing arrow icon. A table titled 'Last Viewed' lists three items: a Resource group last viewed 1 mo ago, a Subscription last viewed 1 mo ago, and a Log Analytics workspace last viewed 1 mo ago. The URL in the address bar is [https://portal.azure.com/?feature.customportal=false#blade/Microsoft\\_Azure\\_HelpSupport门户/HelpSupportBlade](https://portal.azure.com/?feature.customportal=false#blade/Microsoft_Azure_HelpSupport门户/HelpSupportBlade).

2. In **Help + support**, select **New support request**.

The screenshot shows the 'Help + support' section of the Azure portal. At the top, there's a search bar and a back arrow. Below it, the 'Overview' section is selected. Under 'Support', the 'New support request' button is highlighted with a red box. Other options include 'All support requests', 'Support Plans', 'Service Health', 'Advisor', and 'Get started with Azure'.

3. For **Issue type**, select **Service and subscription limits (quotas)**.

The screenshot shows the 'Basics' tab of the support request form. It includes tabs for 'Basics', 'Solutions', 'Details', and 'Review + create'. A note says: 'Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues. Complete the Basics tab by selecting the options that best describe your problem. Providing detailed, accurate information can help to solve your issues faster.' The 'Issue type' field is marked with an asterisk and has a dropdown menu. The menu items are: 'Select an issue type' (highlighted with a blue border), 'Billing', 'Service and subscription limits (quotas)' (highlighted with a red box), 'Subscription management', and 'Technical'. At the bottom of the form, there's a 'Next: Solutions >>' button.

4. For **Subscription**, select the subscription whose quota you want to increase.

Basics      Solutions      Details      Review + create

Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues.

Complete the Basics tab by selecting the options that best describe your problem. Providing detailed, accurate information can help to solve your issues faster.

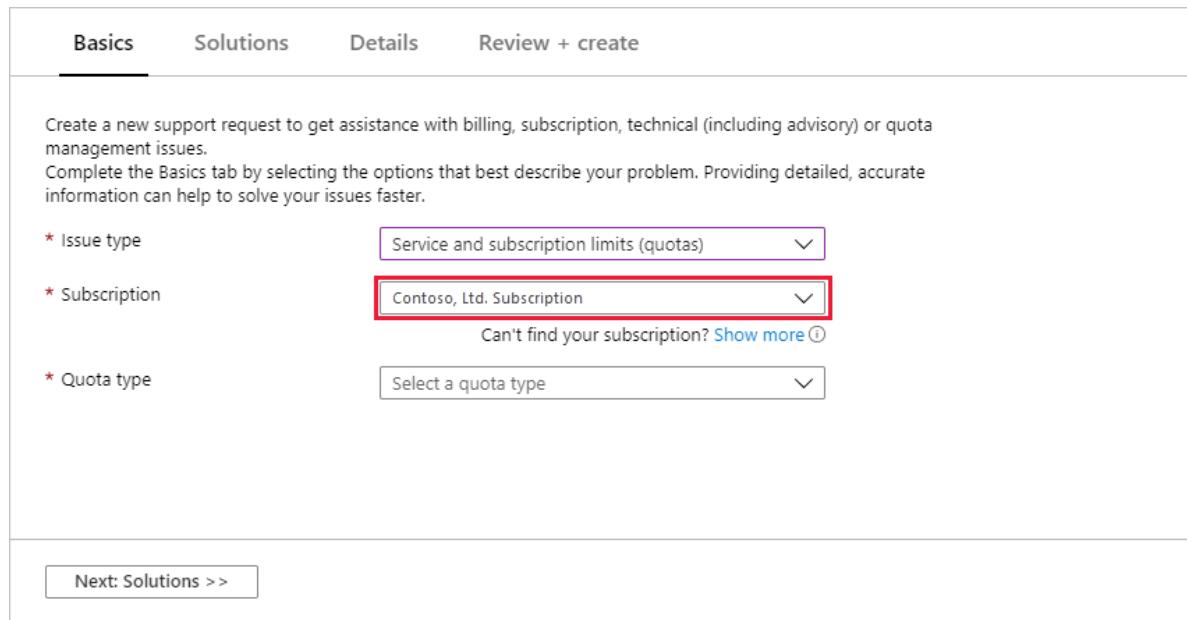
\* Issue type: Service and subscription limits (quotas)

\* Subscription: Contoso, Ltd. Subscription

Can't find your subscription? [Show more](#)

\* Quota type: Select a quota type

[Next: Solutions >>](#)



5. For **Quota type**, select one of the following quota types:

- **SQL Database** for single database and elastic pool quotas.
- **SQL Database Managed Instance** for managed instances.

Then select **Next: Solutions >>**.

Basics      Solutions      Details

Create a new support request to get assistance with management issues.

Complete the Basics tab by selecting the options that information can help to solve your issues faster.

\* Issue type

\* Subscription

\* Quota type: Select a quota type

Other Requests

Search

Service Bus

SQL Data Warehouse

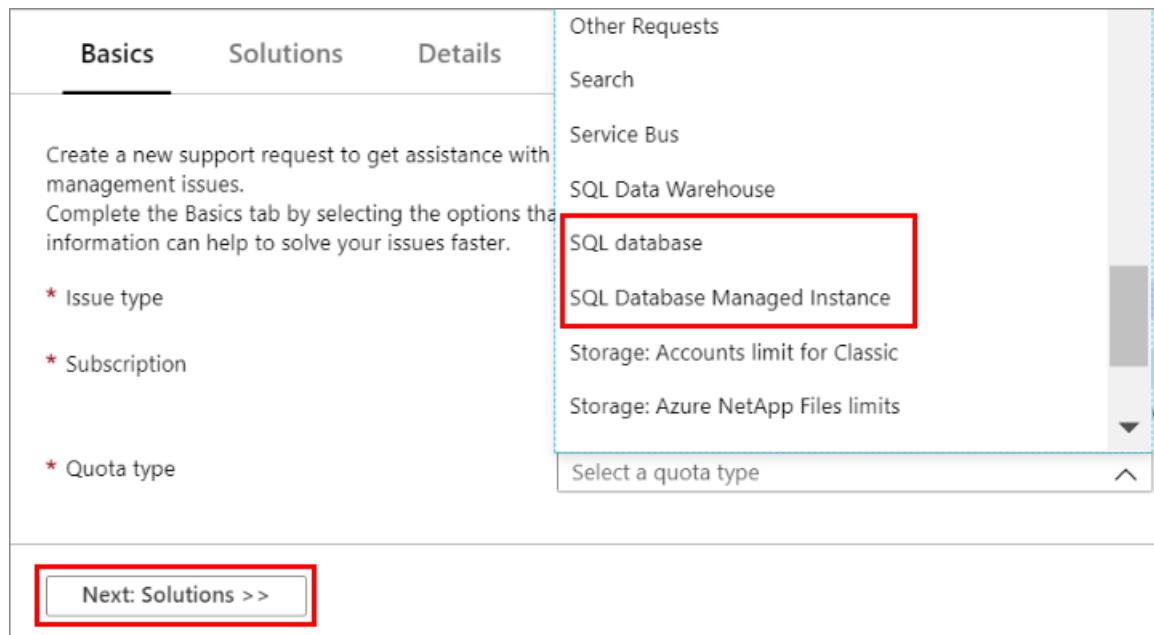
SQL database

SQL Database Managed Instance

Storage: Accounts limit for Classic

Storage: Azure NetApp Files limits

[Next: Solutions >>](#)



6. In the **Details** window, select **Provide details** to enter additional information.

Basics   Solutions   Details   Review + create

Information provided on this tab will be used to further assess your issue and help the support engineer troubleshoot the problem. Verify the contact information before moving to the Review + Create.

**PROBLEM DETAILS**

Additional information is required to promptly process your request for a quota increase.  
\* Provide details

Request Summary      New Limit

**SUPPORT METHOD**

Support plan

<< Previous: Basics      Next: Review + create >>

This screenshot shows the 'Details' tab of a support ticket. It includes sections for 'PROBLEM DETAILS' (with a red box around the 'Provide details' link), 'Request Summary', 'New Limit', 'SUPPORT METHOD', and 'Support plan'. Navigation buttons at the bottom allow switching between 'Basics' and 'Review + create'.

Clicking **Provide details** displays the **Quota details** window that allows you to add additional information. The following sections describe the different options for **SQL Database** and **SQL Database Managed Instance** quota types.

## SQL Database quota types

The following sections describe three quota increase options for the **SQL Database** quota types:

- Database transaction units (DTUs) per server
- Servers per subscription
- Enable subscription access to a region

### Database transaction units (DTUs) per server

Use the following steps to request an increase in the DTUs per server.

1. Select the **Database transaction units (DTUs) per server** quota type.
2. In the **Resource** list, select the resource to target.
3. In the **New quota** field, enter the new DTU limit that you are requesting.

Quota details

SQL database quota type \*

Database transaction units (DTUs) per server

Resource \*

Current Quota Limit For Server  
54000

Current Used Quota for Server  
200

New quota \*

60000

This screenshot shows the 'Quota details' dialog box. It has fields for 'SQL database quota type' (set to 'Database transaction units (DTUs) per server'), 'Resource' (a dropdown menu highlighted with a red box), 'Current Quota Limit For Server' (54000), 'Current Used Quota for Server' (200), and 'New quota' (60000). A green checkmark icon is visible next to the new quota value.

For more information, see [Resource limits for single databases using the DTU purchasing model](#) and [Resources](#)

limits for elastic pools using the DTU purchasing model.

## Servers per subscription

Use the following steps to request an increase in the number of servers per subscription.

1. Select the **Servers per subscription** quota type.
2. In the **Location** list, select the Azure region to use. The quota is per subscription in each region.
3. In the **New quota** field, enter your request for the maximum number of servers in that region.

The screenshot shows the 'Quota details' dialog box. It has a title bar 'Quota details' and a close button 'X'. The first section is 'SQL database quota type \*' with a dropdown menu where 'Servers per subscription' is selected. The second section is 'Location \*' with a dropdown menu where 'East US 2' is selected. Below these are two informational fields: 'Current Server Count Limit' (150) and 'Current Servers Used' (0). The third section is 'New quota \*' with a text input field containing '160', which has a green checkmark icon to its right. The entire 'Location \*' and 'New quota \*' sections are highlighted with a red rectangular border.

For more information, see [SQL Database resource limits and resource governance](#).

## Enable subscription access to a region

Some offer types are not available in every region. You may see an error such as the following:

This location is not available for subscription

If your subscription needs access in a particular region, please use the **Other quota request** option to request access. In your request, specify the offering and SKU details that you want to enable for the region. To explore the offering and SKU options, see [Azure SQL Database pricing](#).

The screenshot shows the 'Quota details' dialog box. It has a title bar 'Quota details' and a close button 'X'. The first section is 'SQL database quota type \*' with a dropdown menu where 'Other quota request' is selected. The second section is 'Description \*' with a text input field containing 'Requesting access for Gen 5 general purpose 16 vCore databases in <region\_name>. Also requesting 3 basic 300 eDTU pools.' A green checkmark icon is visible next to the text. The 'Description \*' section is highlighted with a blue rectangular border.

## Managed instance quota type

For the **SQL Server Managed Instance** quota type, use the following steps:

1. In the **Region** list, select the Azure region to target.

2. Enter the new limits you are requesting for **Subnet** and **vCore**.

**Quota details**

Region \* (Asia Pacific) Australia Central

Please enter the new limits you are requesting:

Resource	Usage	Limit	New limit
Subnet	0	0	
vCore	0	0	

**i** Business Critical (BC) service tier requires four (4) times more vCore capacity than General Purpose (GP) service tier. For example: 1 GP vCore = 1 vCore unit and 1 BC vCore = 4 vCore units. [Learn more](#)

For more information, see [Overview Azure SQL Database managed instance resource limits](#).

## Submit your request

The final step is to fill in the remaining details of your SQL Database quota request. Then select **Next: Review + create>>**, and after reviewing the request details, click **Create** to submit the request.

## Next steps

After you submit your request, it will be reviewed. You will be contacted with an answer based on the information you provided in the form.

For more information about other Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).

# What is a single database in Azure SQL Database

11/7/2019 • 3 minutes to read • [Edit Online](#)

The single database deployment option creates a database in Azure SQL Database with its own set of resources and is managed via a SQL Database server. With a single database, each database is isolated from each other and portable, each with its own service tier within the [DTU-based purchasing model](#) or [vCore-based purchasing model](#) and a guaranteed compute size.

## IMPORTANT

Single database is one of three deployment options for Azure SQL Database. The other two are [elastic pools](#) and [managed instance](#).

## NOTE

For a glossary of terms in Azure SQL Database, see [SQL Database terms glossary](#)

## Dynamic scalability

You can build your first app on a small, single database at low cost in the serverless compute tier or a small compute size in the provisioned compute tier. You change the [compute or service tier](#) manually or programmatically at any time to meet the needs of your solution. You can adjust performance without downtime to your app or to your customers. Dynamic scalability enables your database to transparently respond to rapidly changing resource requirements and enables you to only pay for the resources that you need when you need them.

## Single databases and elastic pools

A single database can be moved into or out of an [elastic pool](#) for resource sharing. For many businesses and applications, being able to create single databases and dial performance up or down on demand is enough, especially if usage patterns are relatively predictable. But if you have unpredictable usage patterns, it can make it hard to manage costs and your business model. Elastic pools are designed to solve this problem. The concept is simple. You allocate performance resources to a pool rather than an individual database and pay for the collective performance resources of the pool rather than for single database performance.

## Monitoring and alerting

You use the built-in [performance monitoring](#) and [alerting tools](#), combined with the performance ratings. Using these tools, you can quickly assess the impact of scaling up or down based on your current or project performance needs. Additionally, SQL Database can [emit metrics and diagnostic logs](#) for easier monitoring.

## Availability capabilities

Single databases, elastic pools, and managed instances all provide many availability characteristics. For information, see [Availability characteristics](#).

## Transact-SQL differences

Most Transact-SQL features that applications use are fully supported in both Microsoft SQL Server and Azure SQL Database. For example, the core SQL components such as data types, operators, string, arithmetic, logical, and cursor functions, work identically in SQL Server and SQL Database. There are, however, a few T-SQL differences in DDL (data-definition language) and DML (data manipulation language) elements resulting in T-SQL statements and queries that are only partially supported (which we discuss later in this article). In addition, there are some features and syntax that is not supported at all because Azure SQL Database is designed to isolate features from dependencies on the master database and the operating system. As such, most server-level activities are inappropriate for SQL Database. T-SQL statements and options are not available if they configure server-level options, operating system components, or specify file system configuration. When such capabilities are required, an appropriate alternative is often available in some other way from SQL Database or from another Azure feature or service.

For more information, see [Resolving Transact-SQL differences during migration to SQL Database](#).

## Security

SQL Database provides a range of [built-in security and compliance](#) features to help your application meet various security and compliance requirements.

### IMPORTANT

Azure SQL Database (all deployment options), has been certified against a number of compliance standards. For more information, see the [Microsoft Azure Trust Center](#) where you can find the most current list of SQL Database compliance certifications.

## Next steps

- To quickly get started with a single database, start with the [Single database quickstart guide](#).
- To learn about migrating a SQL Server database to Azure, see [Migrate to Azure SQL Database](#).
- For information about supported features, see [Features](#).

# Elastic pools help you manage and scale multiple Azure SQL databases

12/6/2019 • 13 minutes to read • [Edit Online](#)

SQL Database elastic pools are a simple, cost-effective solution for managing and scaling multiple databases that have varying and unpredictable usage demands. The databases in an elastic pool are on a single Azure SQL Database server and share a set number of resources at a set price. Elastic pools in Azure SQL Database enable SaaS developers to optimize the price performance for a group of databases within a prescribed budget while delivering performance elasticity for each database.

## What are SQL elastic pools

SaaS developers build applications on top of large scale data-tiers consisting of multiple databases. A common application pattern is to provision a single database for each customer. But different customers often have varying and unpredictable usage patterns, and it's difficult to predict the resource requirements of each individual database user. Traditionally, you had two options:

- Over-provision resources based on peak usage and over pay, or
- Under-provision to save cost, at the expense of performance and customer satisfaction during peaks.

Elastic pools solve this problem by ensuring that databases get the performance resources they need when they need it. They provide a simple resource allocation mechanism within a predictable budget. To learn more about design patterns for SaaS applications using elastic pools, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).

### IMPORTANT

There is no per-database charge for elastic pools. You are billed for each hour a pool exists at the highest eDTU or vCores, regardless of usage or whether the pool was active for less than an hour.

Elastic pools enable the developer to purchase resources for a pool shared by multiple databases to accommodate unpredictable periods of usage by individual databases. You can configure resources for the pool based either on the [DTU-based purchasing model](#) or the [vCore-based purchasing model](#). The resource requirement for a pool is determined by the aggregate utilization of its databases. The amount of resources available to the pool is controlled by the developer budget. The developer simply adds databases to the pool, sets the minimum and maximum resources for the databases (either minimum and maximum DTUs or minimum or maximum vCores depending on your choice of resourcing model), and then sets the resources of the pool based on their budget. A developer can use pools to seamlessly grow their service from a lean startup to a mature business at ever-increasing scale.

Within the pool, individual databases are given the flexibility to auto-scale within set parameters. Under heavy load, a database can consume more resources to meet demand. Databases under light loads consume less, and databases under no load consume no resources. Provisioning resources for the entire pool rather than for single databases simplifies your management tasks. Plus, you have a predictable budget for the pool. Additional resources can be added to an existing pool with no database downtime, except that the databases may need to be moved to provide the additional compute resources for the new

eDTU reservation. Similarly, if extra resources are no longer needed they can be removed from an existing pool at any point in time. And you can add or subtract databases to the pool. If a database is predictably under-utilizing resources, move it out.

**NOTE**

When moving databases into or out of an elastic pool, there is no downtime except for a brief period of time (on the order of seconds) at the end of the operation when database connections are dropped.

## When should you consider a SQL Database elastic pool

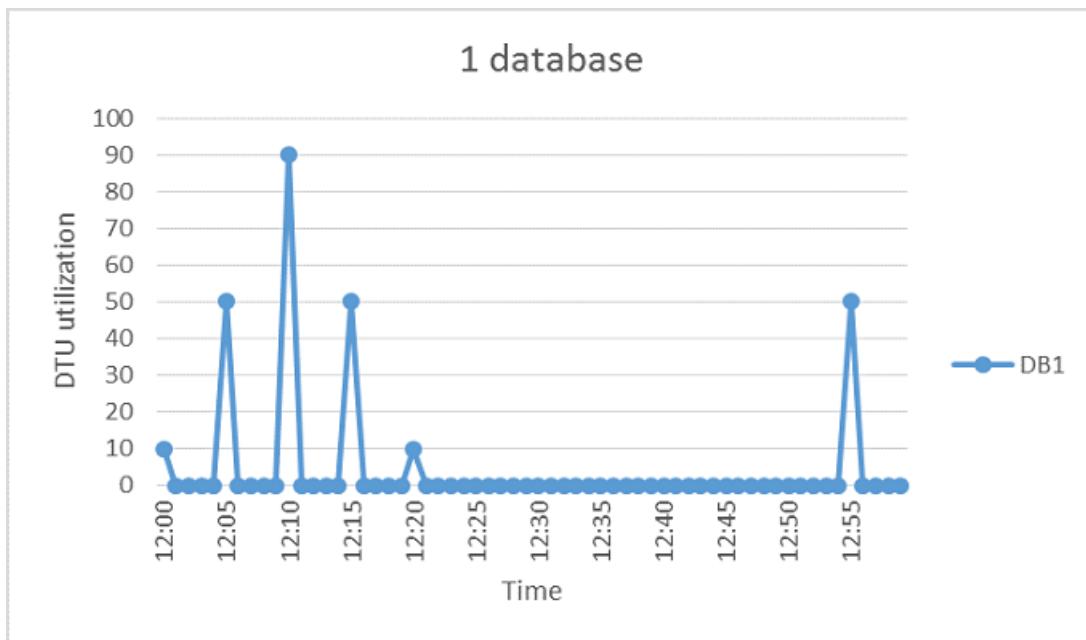
Pools are well suited for a large number of databases with specific utilization patterns. For a given database, this pattern is characterized by low average utilization with relatively infrequent utilization spikes.

The more databases you can add to a pool the greater your savings become. Depending on your application utilization pattern, it's possible to see savings with as few as two S3 databases.

The following sections help you understand how to assess if your specific collection of databases can benefit from being in a pool. The examples use Standard pools but the same principles also apply to Basic and Premium pools.

### Assessing database utilization patterns

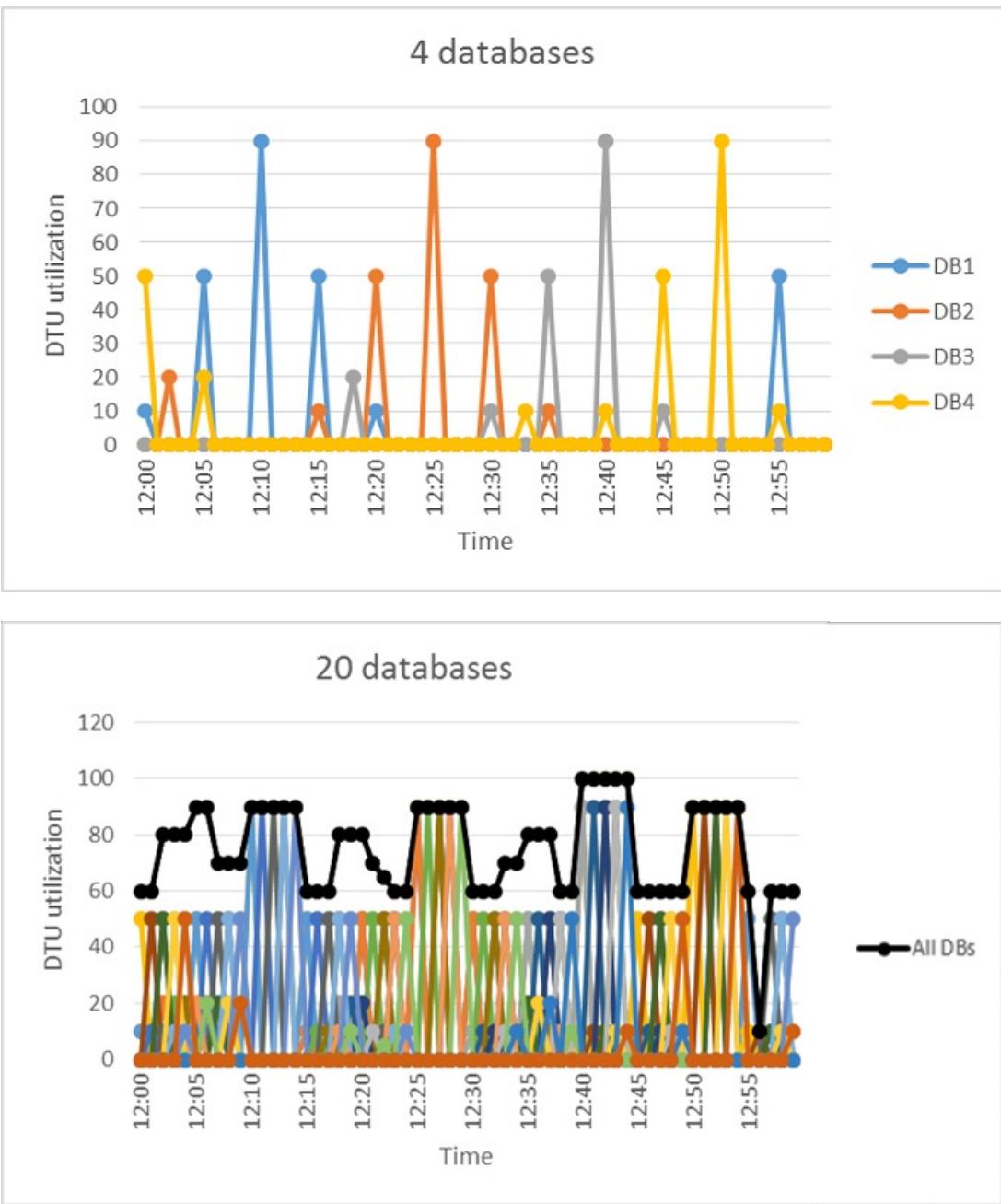
The following figure shows an example of a database that spends much time idle, but also periodically spikes with activity. This is a utilization pattern that is suited for a pool:



For the five-minute period illustrated, DB1 peaks up to 90 DTUs, but its overall average usage is less than five DTUs. An S3 compute size is required to run this workload in a single database, but this leaves most of the resources unused during periods of low activity.

A pool allows these unused DTUs to be shared across multiple databases, and so reduces the DTUs needed and overall cost.

Building on the previous example, suppose there are additional databases with similar utilization patterns as DB1. In the next two figures below, the utilization of four databases and 20 databases are layered onto the same graph to illustrate the non-overlapping nature of their utilization over time using the DTU-based purchasing model:



The aggregate DTU utilization across all 20 databases is illustrated by the black line in the preceding figure. This shows that the aggregate DTU utilization never exceeds 100 DTUs, and indicates that the 20 databases can share 100 eDTUs over this time period. This results in a 20x reduction in DTUs and a 13x price reduction compared to placing each of the databases in S3 compute sizes for single databases.

This example is ideal for the following reasons:

- There are large differences between peak utilization and average utilization per database.
- The peak utilization for each database occurs at different points in time.
- eDTUs are shared between many databases.

The price of a pool is a function of the pool eDTUs. While the eDTU unit price for a pool is 1.5x greater than the DTU unit price for a single database, **pool eDTUs can be shared by many databases and fewer total eDTUs are needed**. These distinctions in pricing and eDTU sharing are the basis of the price savings potential that pools can provide.

The following rules of thumb related to database count and database utilization help to ensure that a pool delivers reduced cost compared to using compute sizes for single databases.

#### Minimum number of databases

If the aggregate amount of resources for single databases is more than 1.5x the resources needed for the pool, then an elastic pool is more cost effective.

#### **DTU-based purchasing model example**

At least two S3 databases or at least 15 S0 databases are needed for a 100 eDTU pool to be more cost-effective than using compute sizes for single databases.

#### **Maximum number of concurrently peaking databases**

By sharing resources, not all databases in a pool can simultaneously use resources up to the limit available for single databases. The fewer databases that concurrently peak, the lower the pool resources can be set and the more cost-effective the pool becomes. In general, not more than 2/3 (or 67%) of the databases in the pool should simultaneously peak to their resources limit.

#### **DTU-based purchasing model example**

To reduce costs for three S3 databases in a 200 eDTU pool, at most two of these databases can simultaneously peak in their utilization. Otherwise, if more than two of these four S3 databases simultaneously peak, the pool would have to be sized to more than 200 eDTUs. If the pool is resized to more than 200 eDTUs, more S3 databases would need to be added to the pool to keep costs lower than compute sizes for single databases.

Note this example doesn't consider utilization of other databases in the pool. If all databases have some utilization at any given point in time, then less than 2/3 (or 67%) of the databases can peak simultaneously.

#### **Resource utilization per database**

A large difference between the peak and average utilization of a database indicates prolonged periods of low utilization and short periods of high utilization. This utilization pattern is ideal for sharing resources across databases. A database should be considered for a pool when its peak utilization is about 1.5 times greater than its average utilization.

**DTU-based purchasing model example:** An S3 database that peaks to 100 DTUs and on average uses 67 DTUs or less is a good candidate for sharing eDTUs in a pool. Alternatively, an S1 database that peaks to 20 DTUs and on average uses 13 DTUs or less is a good candidate for a pool.

## **How do I choose the correct pool size**

The best size for a pool depends on the aggregate resources needed for all databases in the pool. This involves determining the following:

- Maximum resources utilized by all databases in the pool (either maximum DTUs or maximum vCores depending on your choice of resourcing model).
- Maximum storage bytes utilized by all databases in the pool.

For available service tiers for each resource model, see the [DTU-based purchasing model](#) or the [vCore-based purchasing model](#).

In cases where you can't use tooling, the following step-by-step can help you estimate whether a pool is more cost-effective than single databases:

1. Estimate the eDTUs or vCores needed for the pool as follows:

For DTU-based purchasing model:  $\text{MAX}(<\text{Total number of DBs} \times \text{average DTU utilization per DB}>,$

$<\text{Number of concurrently peaking DBs} \times \text{Peak DTU utilization per DB}>)$

For vCore-based purchasing model:  $\text{MAX}(<\text{Total number of DBs} \times \text{average vCore utilization per DB}>,$

$<\text{Number of concurrently peaking DBs} \times \text{Peak vCore utilization per DB}\rangle$

2. Estimate the storage space needed for the pool by adding the number of bytes needed for all the databases in the pool. Then determine the eDTU pool size that provides this amount of storage.
3. For the DTU-based purchasing model, take the larger of the eDTU estimates from Step 1 and Step 2. For the vCore-based purchasing model, take the vCore estimate from Step 1.
4. See the [SQL Database pricing page](#) and find the smallest pool size that is greater than the estimate from Step 3.
5. Compare the pool price from Step 5 to the price of using the appropriate compute sizes for single databases.

## Using other SQL Database features with elastic pools

### Elastic jobs and elastic pools

With a pool, management tasks are simplified by running scripts in [elastic jobs](#). An elastic job eliminates most of tedium associated with large numbers of databases.

For more information about other database tools for working with multiple databases, see [Scaling out with Azure SQL Database](#).

### Business continuity options for databases in an elastic pool

Pooled databases generally support the same [business continuity features](#) that are available to single databases.

- **Point-in-time restore**

Point-in-time restore uses automatic database backups to recover a database in a pool to a specific point in time. See [Point-In-Time Restore](#)

- **Geo-restore**

Geo-restore provides the default recovery option when a database is unavailable because of an incident in the region where the database is hosted. See [Restore an Azure SQL Database or failover to a secondary](#)

- **Active geo-replication**

For applications that have more aggressive recovery requirements than geo-restore can offer, configure [Active geo-replication](#) or an [auto-failover group](#).

## Creating a new SQL Database elastic pool using the Azure portal

There are two ways you can create an elastic pool in the Azure portal.

1. Go to the [Azure portal](#) to create an elastic pool. Search for and select [Azure SQL](#).
2. Select **+Add** to open the **Select SQL deployment option** page. You can view additional information about elastic pools by selecting **Show details** on the **Databases** tile.
3. On the **Databases** tile, select **Elastic pool** in the **Resource type** dropdown, then select **Create**:

The screenshot shows the 'Select SQL deployment option' page in the Azure portal. It lists five service types:

- SQL databases**: Best for modern cloud applications. Hyperscale and serverless options are available. Resource type: Elastic pool. Buttons: Create, Hide details.
- SQL managed instances**: Best for most migrations to the cloud. Lift-and-shift ready. Resource type: Single instance. Buttons: Create, Show details.
- SQL virtual machines**: Best for migrations and applications requiring OS-level access. Lift-and-shift ready. Image: Please select an offer. Buttons: Create, Show details.
- Single database**: Single databases are a great fit for modern, cloud-born applications that need a fully managed database with predictable performance. **Featured capabilities:** Hyperscale storage (up to 100TB), Serverless compute, Easy management.
- Elastic pool**: Elastic pools provide a cost-effective solution for managing the performance of multiple databases with variable usage patterns. **Featured capabilities:** Resource sharing for cost optimization, Simplified performance management.
- Database server**: Database servers are used to manage groups of single databases and elastic pools. **Featured capabilities:** Access management, Backup management, Business continuity management.

4. Or you can create an elastic pool by navigating to an existing Azure SQL server and clicking **+ New pool** to create a pool directly into that server.

#### NOTE

You can create multiple pools on a server, but you can't add databases from different servers into the same pool.

The pool's service tier determines the features available to the elastics in the pool, and the maximum amount of resources available to each database. For details, see [Resource limits for elastic pools in the DTU model](#). For vCore-based resource limits for elastic pools, see [vCore-based resource limits - elastic pools](#).

To configure the resources and pricing of the pool, click **Configure pool**. Then select a service tier, add databases to the pool, and configure the resource limits for the pool and its databases.

When you have completed configuring the pool, you can click 'Apply', name the pool, and click 'OK' to create the pool.

## Monitor an elastic pool and its databases

In the Azure portal, you can monitor the utilization of an elastic pool and the databases within that pool. You can also make a set of changes to your elastic pool and submit all changes at the same time. These changes include adding or removing databases, changing your elastic pool settings, or changing your database settings.

To start monitoring your elastic pool, find and open an elastic pool in the portal. You'll first see a screen that gives you an overview of the status of your elastic pool. This includes:

- Monitoring charts showing resources usage of the elastic pool
- Recent alerts and recommendations, if available, for the elastic pool

The following graphic shows an example elastic pool:

The screenshot shows the 'Overview' blade for a SQL elastic pool named 'RecommendedStandardPool1'. The left sidebar contains links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Configure pool, Locks, Automation script, Alerts (Classic), Database Resource Utilization, and Diagnostics settings. The main area displays resource utilization charts: one for eDTU usage (0%) and another for Storage (53.26% used of 200 GB). A 'Notifications (0)' section indicates no alerts or recommendations. The bottom right shows a 'Recommendations' section.

If you want more information about the pool you can click on any of the available information in this overview. Clicking on the **Resource utilization** chart will take you to the Azure Monitoring view where you can customize the metrics and time window shown in the chart. Clicking on any available notifications will take you to a blade that shows the full details of that alert or recommendation.

If you would like to monitor the databases inside your pool, you can click on **Database resource utilization** in the **Monitoring** section of the resource menu on the left.

The screenshot shows the 'Database Resource Utilization' blade for the same pool. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Configure pool, Locks, Automation script, Alert rules, Database Resource Utilization (selected), and Diagnostics settings. The main area features a chart titled 'eDTU usage for databases for past hour' showing usage for five databases: CRM DATABASE, CRMDB, CRMDB\_2016..., DB01, and DB02. Below the chart is a section to 'Select additional metrics to display below:' with a dropdown set to '0 selected'. A table lists 'Elastic databases within the pool (select up to five)' with columns for DATABASE NAME, AVG EDTU, and PEAK EDTU. All listed databases have a value of 0.

DATABASE NAME	AVG EDTU	PEAK EDTU
CRM Database	0	0
crmdb	0	0
crmdb_2016-11-10T13-18Z	0	0
db01	0	0
db02	0	0

### To customize the chart display

You can edit the chart and the metric page to display other metrics such as CPU percentage, data IO percentage, and log IO percentage used.

On the **Edit Chart** form, you can select a fixed time range or click **custom** to select any 24-hour window in the last two weeks, and then select the resources to monitor.

## To select databases to monitor

By default, the chart in the **Database Resource Utilization** blade will show the top 5 databases by DTU or CPU (depending on your service tier). You can switch up the databases in this chart by selecting and unselecting databases from the list below the chart via the checkboxes on the left.

You can also select more metrics to view side by side in this database table to get a more complete view of your databases performance.

For more information, see [create SQL Database alerts in Azure portal](#).

## Customer case studies

- [SnelStart](#)

SnelStart used elastic pools with Azure SQL Database to rapidly expand its business services at a rate of 1,000 new Azure SQL Databases per month.

- [Umbraco](#)

Umbraco uses elastic pools with Azure SQL Database to quickly provision and scale services for thousands of tenants in the cloud.

- [Daxko/CSI](#)

Daxko/CSI uses elastic pools with Azure SQL Database to accelerate its development cycle and to enhance its customer services and performance.

## Next steps

- To scale elastic pools, see [Scaling elastic pools](#) and [Scale an elastic pool - sample code](#)
- For a video, see [Microsoft Virtual Academy video course on Azure SQL Database elastic capabilities](#)
- To learn more about design patterns for SaaS applications using elastic pools, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).
- For a SaaS tutorial using elastic pools, see [Introduction to the Wingtip SaaS application](#).
- To learn about resource management in elastic pools with many databases, see [Resource management in dense elastic pools](#).

# Azure SQL Database servers and their management

1/3/2020 • 10 minutes to read • [Edit Online](#)

## What is an Azure SQL Database server

A SQL Database server is a logical construct that acts as a central administrative point for multiple single or pooled databases, [logins](#), [firewall rules](#), [auditing rules](#), [threat detection policies](#), and [failover groups](#). A SQL Database server can be in a different region than its resource group. The SQL Database server must exist before you can create the Azure SQL database. All databases managed by a SQL Database server are created within the same region as the SQL Database server.

A SQL Database server is distinct from a SQL Server instance that you may be familiar with in the on-premises world. Specifically, the SQL Database service makes no guarantees regarding location of the databases in relation to the SQL Database server that manages them, and exposes no instance-level access or features. In contrast, the instance databases in a managed instance are all co-located - in the same way that you are familiar with SQL Server in the on-premises world.

When you create a SQL Database server, you provide a server login account and password that has administrative rights to the master database on that server and all databases created on that server. This initial account is a SQL login account. Azure SQL Database supports SQL authentication and Azure Active Directory Authentication for authentication. For information about logins and authentication, see [Managing Databases and Logins in Azure SQL Database](#). Windows Authentication is not supported.

A SQL Database server:

- Is created within an Azure subscription, but can be moved with its contained resources to another subscription
- Is the parent resource for databases, elastic pools, and data warehouses
- Provides a namespace for databases, elastic pools, and data warehouses
- Is a logical container with strong lifetime semantics - delete a server and it deletes the contained databases, elastic pools, and data warehouses
- Participates in [Azure role-based access control \(RBAC\)](#) - databases, elastic pools, and data warehouses within a server inherit access rights from the server
- Is a high-order element of the identity of databases, elastic pools, and data warehouses for Azure resource management purposes (see the URL scheme for databases and pools)
- Collocates resources in a region
- Provides a connection endpoint for database access (`<serverName>.database.windows.net`)
- Provides access to metadata regarding contained resources via DMVs by connecting to a master database
- Provides the scope for management policies that apply to its databases - logins, firewall, audit, threat detection, and such
- Is restricted by a quota within the parent subscription (six servers per subscription by default - see [Subscription limits here](#))
- Provides the scope for database quota and DTU or vCore quota for the resources it contains (such as 45,000 DTU)
- Is the versioning scope for capabilities enabled on contained resources
- Server-level principal logins can manage all databases on a server
- Can contain logins similar to those in instances of SQL Server on your premises that are granted access to one or more databases on the server, and can be granted limited administrative rights. For more information, see [Logins](#).

- The default collation for all databases created on a SQL Database server is `SQL_Latin1_General_CI_AS`, where `Latin1_General` is English (United States), `CP1` is code page 1252, `CI` is case-insensitive, and `AS` is accent-sensitive.

## Manage Azure SQL servers, databases, and firewalls using the Azure portal

You can create the Azure SQL database's resource group ahead of time or while creating the server itself. There are multiple methods for getting to a new SQL server form, either by creating a new SQL server or as part of creating a new database.

### Create a blank SQL Database server

To create an Azure SQL Database server (without a database) using the [Azure portal](#), navigate to a blank SQL server (logical server) form.

### Create a blank or sample SQL database

To create an Azure SQL database using the [Azure portal](#), navigate to a blank SQL Database form and provide the requested information. You can create the Azure SQL database's resource group and SQL Database server ahead of time or while creating the database itself. You can create a blank database or create a sample database based on Adventure Works LT.

The screenshot shows the Azure portal's 'New' blade. On the left, there's a sidebar with a 'Create a resource' button highlighted with a red box. Below it is a list of services: Home, Dashboard, All services, FAVORITES (with All resources, Resource groups, App Services, SQL databases, SQL data warehouses, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center). In the center, there's a search bar labeled 'Search the Marketplace'. Below it, the 'Azure Marketplace' section has tabs for 'See all' and 'Featured'. The 'Featured' tab is selected, showing items like 'Get started', 'Recently created', 'Compute', 'Networking', 'Storage', 'Web', 'Mobile', 'Containers', 'Databases' (which is highlighted with a red box), 'Analytics', 'AI + Machine Learning', 'Internet of Things', 'Integration', 'Security', and 'Identity'. Each item has a thumbnail icon and a link to its 'Quickstart tutorial'.

#### IMPORTANT

For information on selecting the pricing tier for your database, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#).

To create a managed instance, see [Create a managed instance](#)

## Manage an existing SQL server

To manage an existing server, navigate to the server using a number of methods - such as from specific SQL database page, the **SQL servers** page, or the **All resources** page.

To manage an existing database, navigate to the **SQL databases** page and click the database you wish to manage. The following screenshot shows how to begin setting a server-level firewall for a database from the **Overview** page for a database.

The screenshot shows the Azure portal interface for managing a SQL database. On the left, there's a sidebar with various resource group and subscription details. The main area is titled 'Firewall settings' for a server named 'mynewserver-20181208'. It shows a table of rules with one entry: 'ClientIPAddress\_2018-11' with 'START IP' as '73.42.249.7' and 'END IP' as '73.42.249.7'. There are buttons for 'Save' and 'Discard' at the top right. Below the table, it says 'No vnet rules for this server.'

### IMPORTANT

To configure performance properties for a database, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#).

### TIP

For an Azure portal quickstart, see [Create an Azure SQL database in the Azure portal](#).

## Manage Azure SQL servers, databases, and firewalls using PowerShell

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

To create and manage Azure SQL server, databases, and firewalls with Azure PowerShell, use the following PowerShell cmdlets. If you need to install or upgrade PowerShell, see [Install Azure PowerShell module](#). For creating and managing elastic pools, see [Elastic pools](#).

CMDLET	DESCRIPTION
New-AzSqlDatabase	Creates a database
Get-AzSqlDatabase	Gets one or more databases
Set-AzSqlDatabase	Sets properties for a database, or moves an existing database into an elastic pool
Remove-AzSqlDatabase	Removes a database
New-AzResourceGroup	Creates a resource group
New-AzSqlServer	Creates a server
Get-AzSqlServer	Returns information about servers
Set-AzSqlServer	Modifies properties of a server
Remove-AzSqlServer	Removes a server
New-AzSqlServerFirewallRule	Creates a server-level firewall rule
Get-AzSqlServerFirewallRule	Gets firewall rules for a server
Set-AzSqlServerFirewallRule	Modifies a firewall rule in a server
Remove-AzSqlServerFirewallRule	Deletes a firewall rule from a server.
New-AzSqlServerVirtualNetworkRule	Creates a <i>virtual network rule</i> , based on a subnet that is a Virtual Network service endpoint.

#### TIP

For a PowerShell quickstart, see [Create an Azure SQL single database using PowerShell](#). For PowerShell example scripts, see [Use PowerShell to create an Azure SQL single database and configure a firewall rule](#) and [Monitor and scale an Azure SQL single database using PowerShell](#).

## Manage Azure SQL servers, databases, and firewalls using the Azure CLI

To create and manage Azure SQL server, databases, and firewalls with the [Azure CLI](#), use the following [Azure CLI SQL Database](#) commands. Use the [Cloud Shell](#) to run the CLI in your browser, or [install](#) it on macOS, Linux, or Windows. For creating and managing elastic pools, see [Elastic pools](#).

CMDLET	DESCRIPTION
az sql db create	Creates a database
az sql db list	Lists all databases and data warehouses in a server, or all databases in an elastic pool

CMDLET	DESCRIPTION
<a href="#">az sql db list-editions</a>	Lists available service objectives and storage limits
<a href="#">az sql db list-usages</a>	Returns database usages
<a href="#">az sql db show</a>	Gets a database or data warehouse
<a href="#">az sql db update</a>	Updates a database
<a href="#">az sql db delete</a>	Removes a database
<a href="#">az group create</a>	Creates a resource group
<a href="#">az sql server create</a>	Creates a server
<a href="#">az sql server list</a>	Lists servers
<a href="#">az sql server list-usages</a>	Returns server usages
<a href="#">az sql server show</a>	Gets a server
<a href="#">az sql server update</a>	Updates a server
<a href="#">az sql server delete</a>	Deletes a server
<a href="#">az sql server firewall-rule create</a>	Creates a server firewall rule
<a href="#">az sql server firewall-rule list</a>	Lists the firewall rules on a server
<a href="#">az sql server firewall-rule show</a>	Shows the detail of a firewall rule
<a href="#">az sql server firewall-rule update</a>	Updates a firewall rule
<a href="#">az sql server firewall-rule delete</a>	Deletes a firewall rule

#### TIP

For an Azure CLI quickstart, see [Create an Azure SQL single database using the Azure CLI](#). For Azure CLI example scripts, see [Use CLI to create an Azure SQL single database and configure a firewall rule](#) and [Use CLI to monitor and scale an Azure SQL single database](#).

## Manage Azure SQL servers, databases, and firewalls using Transact-SQL

To create and manage Azure SQL server, databases, and firewalls with Transact-SQL, use the following T-SQL commands. You can issue these commands using the Azure portal, [SQL Server Management Studio](#), [Visual Studio Code](#), or any other program that can connect to an Azure SQL Database server and pass Transact-SQL commands. For managing elastic pools, see [Elastic pools](#).

**IMPORTANT**

You cannot create or delete a server using Transact-SQL.

COMMAND	DESCRIPTION
<a href="#">CREATE DATABASE (Azure SQL Database)</a>	Creates a new database. You must be connected to the master database to create a new database.
<a href="#">ALTER DATABASE (Azure SQL Database)</a>	Modifies an Azure SQL database.
<a href="#">ALTER DATABASE (Azure SQL Data Warehouse)</a>	Modifies an Azure SQL Data Warehouse.
<a href="#">DROP DATABASE (Transact-SQL)</a>	Deletes a database.
<a href="#">sys.database_service_objectives (Azure SQL Database)</a>	Returns the edition (service tier), service objective (pricing tier), and elastic pool name, if any, for an Azure SQL database or an Azure SQL Data Warehouse. If logged on to the master database in an Azure SQL Database server, returns information on all databases. For Azure SQL Data Warehouse, you must be connected to the master database.
<a href="#">sys.dm_db_resource_stats (Azure SQL Database)</a>	Returns CPU, IO, and memory consumption for an Azure SQL Database database. One row exists for every 15 seconds, even if there is no activity in the database.
<a href="#">sys.resource_stats (Azure SQL Database)</a>	Returns CPU usage and storage data for an Azure SQL Database. The data is collected and aggregated within five-minute intervals.
<a href="#">sys.database_connection_stats (Azure SQL Database)</a>	Contains statistics for SQL Database database connectivity events, providing an overview of database connection successes and failures.
<a href="#">sys.event_log (Azure SQL Database)</a>	Returns successful Azure SQL Database database connections, connection failures, and deadlocks. You can use this information to track or troubleshoot your database activity with SQL Database.
<a href="#">sp_set_firewall_rule (Azure SQL Database)</a>	Creates or updates the server-level firewall settings for your SQL Database server. This stored procedure is only available in the master database to the server-level principal login. A server-level firewall rule can only be created using Transact-SQL after the first server-level firewall rule has been created by a user with Azure-level permissions
<a href="#">sys.firewall_rules (Azure SQL Database)</a>	Returns information about the server-level firewall settings associated with your Microsoft Azure SQL Database.
<a href="#">sp_delete_firewall_rule (Azure SQL Database)</a>	Removes server-level firewall settings from your SQL Database server. This stored procedure is only available in the master database to the server-level principal login.

COMMAND	DESCRIPTION
<a href="#">sp_set_database_firewall_rule (Azure SQL Database)</a>	Creates or updates the database-level firewall rules for your Azure SQL Database or SQL Data Warehouse. Database firewall rules can be configured for the master database, and for user databases on SQL Database. Database firewall rules are useful when using contained database users.
<a href="#">sys.database_firewall_rules (Azure SQL Database)</a>	Returns information about the database-level firewall settings associated with your Microsoft Azure SQL Database.
<a href="#">sp_delete_database_firewall_rule (Azure SQL Database)</a>	Removes database-level firewall setting from your Azure SQL Database or SQL Data Warehouse.

#### TIP

For a quickstart using SQL Server Management Studio on Microsoft Windows, see [Azure SQL Database: Use SQL Server Management Studio to connect and query data](#). For a quickstart using Visual Studio Code on the macOS, Linux, or Windows, see [Azure SQL Database: Use Visual Studio Code to connect and query data](#).

## Manage Azure SQL servers, databases, and firewalls using the REST API

To create and manage Azure SQL server, databases, and firewalls, use these REST API requests.

COMMAND	DESCRIPTION
<a href="#">Servers - Create or update</a>	Creates or updates a new server.
<a href="#">Servers - Delete</a>	Deletes a SQL server.
<a href="#">Servers - Get</a>	Gets a server.
<a href="#">Servers - List</a>	Returns a list of servers.
<a href="#">Servers - List by resource group</a>	Returns a list of servers in a resource group.
<a href="#">Servers - Update</a>	Updates an existing server.
<a href="#">Databases - Create or update</a>	Creates a new database or updates an existing database.
<a href="#">Databases - Delete</a>	Deletes a database.
<a href="#">Databases - Get</a>	Gets a database.
<a href="#">Databases - List by elastic pool</a>	Returns a list of databases in an elastic pool.
<a href="#">Databases - List by server</a>	Returns a list of databases in a server.
<a href="#">Databases - Update</a>	Updates an existing database.
<a href="#">Firewall rules - Create or update</a>	Creates or updates a firewall rule.

COMMAND	DESCRIPTION
Firewall rules - Delete	Deletes a firewall rule.
Firewall rules - Get	Gets a firewall rule.
Firewall rules - List by server	Returns a list of firewall rules.

## Next steps

- To learn about migrating a SQL Server database to Azure, see [Migrate to Azure SQL Database](#).
- For information about supported features, see [Features](#).

# Quickstart: Create a single database in Azure SQL Database using the Azure portal, PowerShell, and Azure CLI

2/14/2020 • 7 minutes to read • [Edit Online](#)

Creating a [single database](#) is the quickest and simplest deployment option for creating a database in Azure SQL Database. This quickstart shows you how to create and then query a single database using the Azure portal.

If you don't have an Azure subscription, [create a free account](#).

For all steps in this quickstart, sign in to the [Azure portal](#).

## Create a single database

A single database can either be created in the provisioned or serverless compute tier.

- A single database in the provisioned compute tier is pre-allocated a fixed amount of compute resources including CPU and memory using one of two [purchasing models](#).
- A single database in the serverless compute tier has a range of compute resources including CPU and memory that are auto-scaled and is only available in the [vCore-based purchasing models](#).

When you create a single database, you also define a [SQL Database server](#) to manage it and place it within [Azure resource group](#) in a specified region.

### NOTE

This quickstart uses the [vCore-based purchasing model](#), but the [DTU-based purchasing model](#) is also available.

To create a single database containing the AdventureWorksLT sample data:

In this step, you will create an Azure SQL Database single database.

### IMPORTANT

Be sure to set up firewall rules to use the public IP address of the computer you're using to complete this article.

For information, see [Create a database-level firewall rule](#) or to determine the IP address used for the server-level firewall rule for your computer see [Create a server-level firewall](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Create your resource group and single database using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type *Azure SQL* in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.

2. Select **+ Add** to open the **Select SQL deployment option** page. You can view additional information about the different databases by selecting **Show details** on the **Databases** tile.

3. Select **Create**:

The screenshot shows the 'Select SQL deployment option' page. It features three main service options:

- SQL databases**: Best for modern cloud applications. Hyperscale and serverless options are available. Resource type dropdown: Single database. Buttons: Create (highlighted with a red box), Show details.
- SQL managed instances**: Best for most migrations to the cloud. Lift-and-shift ready. Resource type dropdown: Single instance. Buttons: Create, Show details.
- SQL virtual machines**: Best for migrations and applications requiring OS-level access. Lift-and-shift ready. Image dropdown. Buttons: Create, Show details.

4. On the **Basics** tab, in the **Project Details** section, type or select the following values:

- **Subscription**: Drop down and select the correct subscription, if it doesn't appear.
- **Resource group**: Select **Create new**, type `myResourceGroup`, and select **OK**.

The screenshot shows the 'Create SQL Database' Basics tab. It includes the following sections:

- Project details**: Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.
- Subscription \***: A dropdown menu.
- Resource group \***: A dropdown menu with options: Select a resource group (highlighted with a purple border) and Create new.

5. In the **Database Details** section, type or select the following values:

- **Database name**: Enter `mySampleDatabase`.
- **Server**: Select **Create new**, enter the following values and then select **Select**.
  - **Server name**: Type `mysqlserver`; along with some numbers for uniqueness.
  - **Server admin login**: Type `azureuser`.
  - **Password**: Type a complex password that meets password requirements.
  - **Location**: Choose a location from the drop-down, such as `West US`.

New server

\* Server name  
mysqlserver .database.windows.net

\* Server admin login  
azureuser

\* Password  
.....

\* Confirm password  
.....

\* Location  
West US

Allow Azure services to access server

#### IMPORTANT

Remember to record the server admin login and password so you can log in to the server and databases for this and other quickstarts. If you forget your login or password, you can get the login name or reset the password on the **SQL server** page. To open the **SQL server** page, select the server name on the database **Overview** page after database creation.

- **Want to use SQL elastic pool:** Select the **No** option.
- **Compute + storage:** Select **Configure database**.

DATABASE DETAILS

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

\* Database name  
mySampleDatabase

\* Server [\(new\) mysqlserver \( West US \)](#)  
[Create new](#)

\* Want to use SQL elastic pool? [?](#)  Yes  No

\* Compute + storage [?](#)  
Standard S0  
10 DTUs, 1 GB storage  
[Configure database](#)

\* Resource group [?](#)  
(New) myResourceGroup

- Select **Provisioned**.

**Compute tier**

- Provisioned** (selected) 
 Compute resources are pre-allocated  
Billed per hour based on vCores configured
- Serverless** 
 Compute resources are auto-scaled  
Billed per second based on vCores used

**Hardware Configuration**

**Gen5** up to 80 vCores, up to 408 GB memory [Change configuration](#)

**Save money**  
Save up to 55% with a license you already own. Already have a SQL Server license? [○ Yes](#) [No](#)

**vCores** [How do vCores compare with DTUs?](#)  2 vCores

**Data max size**  32 GB  1 TB 32 GB

**Cost summary**

Gen5 - General Purpose (GP_Gen5_2)
Cost per vCore (in USD)
vCores selected <b>x 2</b>
Cost per GB (in USD)
Max storage selected (in GB) <b>x</b>
<b>ESTIMATED COST / MONTH</b>

**9.6 GB LOG SPACE ALLOCATED**

[Apply](#)

- Review the settings for **vCores**, and **Data max size**. Change these as desired.
  - Optionally, you can also select **Change configuration** to change the hardware generation.
- Select **Apply**.

6. Select the **Networking** tab and decide if you want to **Allow Azure services and resources to access this server**, or add a **private endpoint**.

**Create SQL Database**

**Basics** **Networking** [Additional settings](#) [Tags](#) [Review + create](#)

Configure network access and connectivity for your server. The configuration selected below will apply to the selected server 'carlrabperfsrv' and all databases it manages. [Learn more](#)

**Firewall rules**  
The settings displayed below are read-only. They can be modified from the "Firewalls and virtual networks" blade after database creation. [Learn more](#)

Allow Azure services and resources to access this server  No  Yes

**Private endpoints (preview)**  
Private endpoint connections are associated with a private IP address within a Virtual Network. The list below shows all the private endpoint connections for this server. Note that private endpoint connections are defined at the server level and they provide access to all databases in the server. [Learn more](#)

[+ Add private endpoint](#)

Name	Subscription
<i>Click on add to create private endpoint</i>	

7. Select the **Additional settings** tab.

8. In the **Data source** section, under **Use existing data**, select **Sample**.

Home > Azure SQL > Select SQL deployment option > Create SQL Database

## Create SQL Database

Microsoft

Basics Networking Additional settings Tags Review + create

Customize additional configuration parameters including collation & sample data.

**Data source**

Start with a blank database, restore from a backup or select sample data to populate your new database.

Use existing data \*

None Backup Sample

AdventureWorksLT will be created as the sample database.

**Database collation**

Database collation defines the rules that sort and compare data, and cannot be changed after database creation. The default database collation is SQL\_Latin1\_General\_CI\_AS. [Learn more](#)

Collation ⓘ SQL\_Latin1\_General\_CI\_AS

**Advanced data security**

Protect your data using advanced data security, a unified security package including data classification, vulnerability assessment and advanced threat protection for your server. [Learn more](#)

**Advanced data security has already been enabled on the selected server.**

#### IMPORTANT

Make sure to select the **Sample (AdventureWorksLT)** data so you can follow easily this and other Azure SQL Database quickstarts that use this data.

9. Leave the rest of the values as default and select **Review + Create** at the bottom of the form.
10. Review the final settings and select **Create**.
11. On the **SQL Database** form, select **Create** to deploy and provision the resource group, server, and database.

## Query the database

Now that you've created the database, use the built-in query tool in the Azure portal to connect to the database and query the data.

1. On the **SQL Database** page for your database, select **Query editor (preview)** in the left menu.

The screenshot shows the 'mySampleDatabase - Query editor (preview)' interface. On the left, there's a navigation bar with options like Overview, Activity log, Tags, Diagnose and solve problems, Quick start, and Query editor (preview). The 'Query editor (preview)' option is highlighted with a red box. The main area has a large 'SQL' logo and the text 'Welcome to SQL Database Query Editor'. It includes fields for 'Authorization type' (set to 'SQL server authentication'), 'Login' (set to 'azureuser'), and 'password'. There's also an 'Active Directory single sign on' section and a 'Settings' sidebar on the left.

2. Enter your login information, and select **OK**.
3. Enter the following query in the **Query editor** pane.

```
SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
FROM SalesLT.ProductCategory pc
JOIN SalesLT.Product p
ON pc.productcategoryid = p.productcategoryid;
```

4. Select **Run**, and then review the query results in the **Results** pane.

The screenshot shows the 'mySampleDatabase - Query editor (preview)' interface after running the query. The 'Query 1' pane shows the executed SQL code. The 'Results' pane is selected and displays the query results in a table:

CATEGORYNAME	PRODUCTNAME
Road Frames	HL Road Frame - Black, 58
Road Frames	HL Road Frame - Red, 58
Helmets	Sport-100 Helmet, Red
Helmets	Sport-100 Helmet, Black
Socks	Mountain Bike Socks, M
Socks	Mountain Bike Socks, L
Helmets	Sport-100 Helmet, Blue
Cans	AWC Logo Can

5. Close the **Query editor** page, and select **OK** when prompted to discard your unsaved edits.

## Clean up resources

Keep this resource group, database server, and single database if you want to go to the [Next steps](#). The next steps show you how to connect and query your database using different methods.

When you're finished using these resources, you can delete them as follows:

1. From the left menu in the Azure portal, select **Resource groups**, and then select **myResourceGroup**.
2. On your resource group page, select **Delete resource group**.
3. Enter *myResourceGroup* in the field, and then select **Delete**.

## Next steps

- Create a server-level firewall rule to connect to the single database from on-premises or remote tools.

For more information, see [Create a server-level firewall rule](#).

- After you create a server-level firewall rule, [connect and query](#) your database using several different tools and languages.
  - [Connect and query using SQL Server Management Studio](#)
  - [Connect and query using Azure Data Studio](#)
- To create a single database in the provisioned compute tier using Azure CLI, see [Azure CLI samples](#).
- To create a single database in the provisioned compute tier using Azure PowerShell, see [Azure PowerShell samples](#).
- To create a single database in the serverless compute tier using Azure Powershell, see [Create serverless database](#).

# Quickstart: Create a single database in Azure SQL Database using the Azure Resource Manager template

12/23/2019 • 3 minutes to read • [Edit Online](#)

Creating a [single database](#) is the quickest and simplest deployment option for creating a database in Azure SQL Database. This quickstart shows you how to create a single database using the Azure Resource Manager template. For more information, see [Azure Resource Manager documentation](#).

If you don't have an Azure subscription, [create a free account](#).

## Create a single database

A single database has a defined set of compute, memory, IO, and storage resources using one of two [purchasing models](#). When you create a single database, you also define a [SQL Database server](#) to manage it and place it within [Azure resource group](#) in a specified region.

The following JSON file is the template that is used in this article. The template is stored in [GitHub](#). More Azure SQL database template samples can be found in [Azure Quickstart Templates](#).

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 " projectName": {
 "type": "string",
 "metadata": {
 "description": "Specify a project name that is used to generate resource names."
 }
 },
 " location": {
 "type": "string",
 "defaultValue": "[resourceGroup().location]",
 "metadata": {
 "description": "Specify the Azure location for all services."
 }
 },
 " adminUser": {
 "type": "string",
 "metadata": {
 "description": "Specify the username for the database server administrator."
 }
 },
 " adminPassword": {
 "type": "securestring",
 "metadata": {
 "description": "Specify the password for the database server administrator."
 }
 }
 },
 "variables": {
 " databaseServerName": "[concat(toLower(parameters('projectName')), 'server')]",
 " databaseName": "[concat(parameters('projectName'), 'db')]"
 },
 "resources": [
 {
 "type": "Microsoft.Sql/servers",
 "name": "[variables('databaseServerName')]",
 "apiVersion": "2014-04-01",
 "properties": {
 "location": "[parameters('location')]",
 "administratorLogin": "[parameters('adminUser')]",
 "administratorLoginPassword": "[parameters('adminPassword')]"
 }
 },
 {
 "type": "Microsoft.Sql/databases",
 "name": "[concat(variables('databaseServerName'), '/', variables('databaseName'))]",
 "apiVersion": "2014-04-01",
 "dependsOn": "[resourceId('Microsoft.Sql/servers', variables('databaseServerName'))]",
 "properties": {
 "collation": "Latin1_General_CI_AS",
 "maxSizeBytes": 1073741824,
 "minSizeBytes": 1073741824
 }
 }
]
}
```

```

 "name": "[variables('databaseServerName')]",
 "location": "[parameters('location')]",
 "apiVersion": "2015-05-01-preview",
 "properties": {
 "administratorLogin": "[parameters('adminUser')]",
 "administratorLoginPassword": "[parameters('adminPassword')]",
 "version": "12.0"
 },
 "resources": [
 {
 "type": "Microsoft.Sql/servers/databases",
 "name": "[concat(string(variables('databaseServerName')), '/', string(variables('databaseName')))]",
 "location": "[parameters('location')]",
 "apiVersion": "2017-10-01-preview",
 "dependsOn": [
 "[resourceID('Microsoft.Sql/servers/', variables('databaseServerName'))]"
]
 },
 {
 "type": "firewallrules",
 "name": "AllowAllAzureIps",
 "location": "[parameters('location')]",
 "apiVersion": "2015-05-01-preview",
 "dependsOn": [
 "[resourceID('Microsoft.Sql/servers/', variables('databaseServerName'))]"
],
 "properties": {
 "startIpAddress": "0.0.0.0",
 "endIpAddress": "0.0.0.0"
 }
 }
]
}

```

Select **Try it** from the following PowerShell code block to open Azure Cloud Shell.

- [PowerShell](#)
- [Azure CLI](#)

```

$ projectName = Read-Host -Prompt "Enter a project name that is used for generating resource names"
$ location = Read-Host -Prompt "Enter an Azure location (i.e. centralus)"
$ adminUser = Read-Host -Prompt "Enter the SQL server administrator username"
$ adminPassword = Read-Host -Prompt "Enter the SQL server administrator password" -AsSecureString

$ resourceGroupName = "${projectName}rg"

New-AzResourceGroup -Name $ resourceGroupName -Location $ location
New-AzResourceGroupDeployment -ResourceGroupName $ resourceGroupName -TemplateFile "D:\GitHub\azure-docs-json-samples\SQLServerAndDatabase\azuredeploy.json" - projectName $ projectName -adminUser $ adminUser -adminPassword $ adminPassword

Read-Host -Prompt "Press [ENTER] to continue ..."

```

## Query the database

To query the database, see [Query the database](#).

## Clean up resources

Keep this resource group, database server, and single database if you want to go to the [Next steps](#). The next steps

show you how to connect and query your database using different methods.

To delete the resource group:

- [PowerShell](#)
- [Azure CLI](#)

```
$resourceGroupName = Read-Host -Prompt "Enter the Resource Group name"
Remove-AzResourceGroup -Name $resourceGroupName
```

## Next steps

- Create a server-level firewall rule to connect to the single database from on-premises or remote tools. For more information, see [Create a server-level firewall rule](#).
- After you create a server-level firewall rule, [connect and query](#) your database using several different tools and languages.
  - [Connect and query using SQL Server Management Studio](#)
  - [Connect and query using Azure Data Studio](#)
- To create a single database using Azure CLI, see [Azure CLI samples](#).
- To create a single database using Azure PowerShell, see [Azure PowerShell samples](#).

# Quickstart: Create a server-level firewall rule for single and pooled databases using the Azure portal

11/7/2019 • 3 minutes to read • [Edit Online](#)

This quickstart walks through how to create a [server-level firewall rule](#) for single and pooled databases in Azure SQL Database using the Azure portal to enable you to connect to database servers, single databases, and elastic pools and their databases. A firewall rule is required to connect from other Azure resources and from on-premises resources.

## Prerequisites

This quickstart uses the resources created in [Create a single database using the Azure portal](#) as its starting point.

## Sign in to the Azure portal

Sign in to the [Azure portal](#).

## Create a server-level IP firewall rule

The SQL Database service creates a firewall at the database server level for single and pooled databases. This firewall prevents client applications from connecting to the server or any of its single or pooled databases unless you create an IP firewall rule to open the firewall. For a connection from an IP address outside Azure, create a firewall rule for a specific IP address or range of addresses that you want to be able to connect. For more information about server-level and database-level IP firewall rules, see [SQL Database server-level and database-level IP firewall rules](#).

### NOTE

SQL Database communicates over port 1433. If you're trying to connect from within a corporate network, outbound traffic over port 1433 might not be allowed by your network's firewall. If so, you can't connect to your Azure SQL Database server unless your IT department opens port 1433.

### IMPORTANT

A firewall rule of 0.0.0.0 enables all Azure services to pass through the server-level firewall rule and attempt to connect to a single or pooled database through the server.

Follow these steps to create a server-level IP firewall rule for your client's IP address and enable external connectivity through the SQL Database firewall for your IP address only.

1. After the [prerequisite Azure SQL database](#) deployment completes, select **SQL databases** from the left-hand menu and then choose **mySampleDatabase** on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified server name (such as **mynewserver-20170824.database.windows.net**) and provides options for further configuration.
2. Copy this fully qualified server name to use when connecting to your server and its databases in other

quickstarts.

Microsoft Azure

Home > SQL databases > mySampleDatabase

SQL databases

mySampleDatabase

Overview

Activity log

Tags

Diagnose and solve problems

Quick start

Query editor (preview)

Resource group (change)  
myResourceGroup

Status  
Online

Location  
East US

Subscription (change)

Pricing tier  
Standard S0: 10 DTUs

Subscription ID

Server name  
mynewserver-20181208.database.windows.net

Elastic pool  
No elastic pool

Connection strings  
Show database connection strings

Oldest restore point  
2018-11-28 23:14 UTC

3. Select **Set server firewall** on the toolbar. The **Firewall settings** page for the database server opens.

Microsoft Azure

Home > SQL databases > mySampleDatabase > Firewall settings

Copy, Restore, Export, Set server firewall, Save, Discard, Add client IP

Resource group (change)  
myResourceGroup

Status  
Online

Location  
East US

Subscription (change)

Subscription ID

Tags (change)  
Click here to add tags

Allow access to Azure services  
ON

Client IP address  
73.42.249.7

RULE NAME  
ClientIPAddress\_2018-11

START IP  
73.42.249.7

END IP  
73.42.249.7

Virtual networks  
+ Add existing virtual network  
+ Create new virtual network

Resource utilization (mySampleDatabase)

1 hour, 24 hours, 7 days, View: Max

4. Choose **Add client IP** on the toolbar to add your current IP address to a new server-level IP firewall rule. A server-level IP firewall rule can open port 1433 for a single IP address or a range of IP addresses.

#### IMPORTANT

By default, access through the SQL Database firewall is disabled for all Azure services. Choose **ON** on this page if you want to enable access for all Azure services.

5. Select **Save**. A server-level IP firewall rule is created for your current IP address opening port 1433 on the SQL Database server.
6. Close the **Firewall settings** page.

Using SQL Server Management Studio or another tool of your choice, you can now connect to the SQL Database server and its databases from this IP address using the server admin account created previously.

## Clean up resources

Save these resources if you want to go to [Next steps](#) and learn how to connect and query your database using a number of different methods. If, however, you want to delete the resources that you created in this quickstart, use the following steps.

1. From the left-hand menu in the Azure portal, select **Resource groups** and then select **myResourceGroup**.
2. On your resource group page, select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

## Next steps

- Now that you have a database, you can [connect and query](#) using one of your favorite tools or languages, including
  - [Connect and query using SQL Server Management Studio](#)
  - [Connect and query using Azure Data Studio](#)
- To learn how to design your first database, create tables, and insert data, see one of these tutorials:
  - [Design your first single database in Azure SQL Database using SSMS](#)
  - [Design a single database in Azure SQL Database and connect with C# and ADO.NET](#)

# Tutorial: Migrate SQL Server to a single database or pooled database in Azure SQL Database offline using DMS

1/8/2020 • 12 minutes to read • [Edit Online](#)

You can use Azure Database Migration Service to migrate the databases from an on-premises SQL Server instance to [Azure SQL Database](#). In this tutorial, you migrate the **Adventureworks2012** database restored to an on-premises instance of SQL Server 2016 (or later) to a single database or pooled database in Azure SQL Database by using Azure Database Migration Service.

In this tutorial, you learn how to:

- Assess your on-premises database by using the Data Migration Assistant.
- Migrate the sample schema by using the Data Migration Assistant.
- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.
- Download a migration report.

## TIP

When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article describes an offline migration from SQL Server to a single database or pooled database in Azure SQL Database. For an online migration, see [Migrate SQL Server to Azure SQL Database online using DMS](#).

## Prerequisites

To complete this tutorial, you need to:

- Download and install [SQL Server 2016 or later](#).
- Enable the TCP/IP protocol, which is disabled by default during SQL Server Express installation, by following the instructions in the article [Enable or Disable a Server Network Protocol](#).
- Create a single (or pooled) database in Azure SQL Database, which you do by following the detail in the article [Create a single database in Azure SQL Database using the Azure portal](#).

## NOTE

If you use SQL Server Integration Services (SSIS) and want to migrate the catalog database for your SSIS projects/packages (SSISDB) from SQL Server to Azure SQL Database, the destination SSISDB will be created and managed automatically on your behalf when you provision SSIS in Azure Data Factory (ADF). For more information about migrating SSIS packages, see the article [Migrate SQL Server Integration Services packages to Azure](#).

- Download and install the [Data Migration Assistant](#) v3.3 or later.
- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either [ExpressRoute](#) or [VPN](#). For more information about creating a virtual network, see the [Virtual Network Documentation](#), and especially the quickstart articles with step-by-step details.

#### NOTE

During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:

- Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
- Storage endpoint
- Service bus endpoint

This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

If you don't have site-to-site connectivity between the on-premises network and Azure or if there is limited site-to-site connectivity bandwidth, consider using Azure Database Migration Service in hybrid mode (Preview). Hybrid mode leverages an on-premises migration worker together with an instance of Azure Database Migration Service running in the cloud. To create an instance of Azure Database Migration Service in hybrid mode, see the article [Create an instance of Azure Database Migration Service in hybrid mode using the Azure portal](#).

- Ensure that your virtual network Network Security Group rules don't block the following inbound communication ports to Azure Database Migration Service: 443, 53, 9354, 445, 12000. For more detail on Azure virtual network NSG traffic filtering, see the article [Filter network traffic with network security groups](#).
- Configure your [Windows Firewall for database engine access](#).
- Open your Windows firewall to allow Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433.
- If you're running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that Azure Database Migration Service can connect to a named instance on your source server.
- When using a firewall appliance in front of your source database(s), you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level IP [firewall rule](#) for the Azure SQL Database server to allow Azure Database Migration Service access to the target databases. Provide the subnet range of the virtual network used for Azure Database Migration Service.
- Ensure that the credentials used to connect to source SQL Server instance have [CONTROL SERVER](#) permissions.
- Ensure that the credentials used to connect to target Azure SQL Database instance have [CONTROL DATABASE](#) permission on the target Azure SQL databases.

## Assess your on-premises database

Before you can migrate data from an on-premises SQL Server instance to a single database or pooled database in Azure SQL Database, you need to assess the SQL Server database for any blocking issues that might prevent migration. Using the Data Migration Assistant v3.3 or later, follow the steps described in the article [Performing a SQL Server migration assessment](#) to complete the on-premises database assessment. A summary of the required steps follows:

1. In the Data Migration Assistant, select the New (+) icon, and then select the **Assessment** project type.
2. Specify a project name, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database**, and then select **Create** to create the project.

When you're assessing the source SQL Server database migrating to a single database or pooled database in Azure SQL Database, you can choose one or both of the following assessment report types:

- Check database compatibility
- Check feature parity

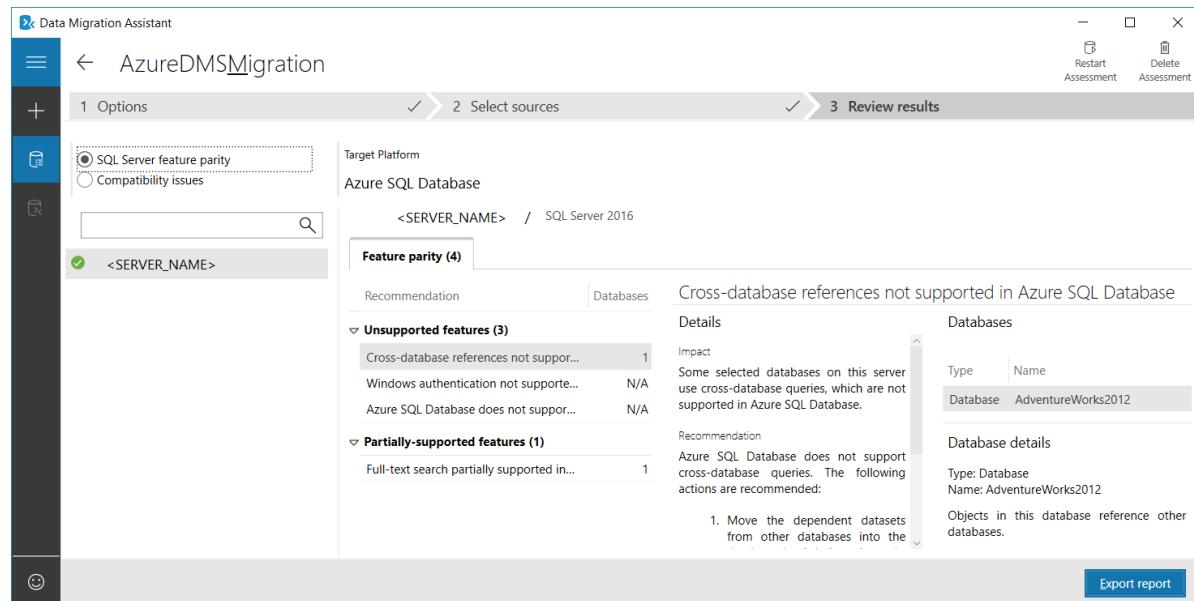
Both report types are selected by default.

3. In the Data Migration Assistant, on the **Options** screen, select **Next**.
4. On the **Select sources** screen, in the **Connect to a server** dialog box, provide the connection details to your SQL Server, and then select **Connect**.
5. In the **Add sources** dialog box, select **AdventureWorks2012**, select **Add**, and then select **Start Assessment**.

#### NOTE

If you use SSIS, DMA does not currently support the assessment of the source SSISDB. However, SSIS projects/packages will be assessed/validated as they are redeployed to the destination SSISDB hosted by Azure SQL Database. For more information about migrating SSIS packages, see the article [Migrate SQL Server Integration Services packages to Azure](#).

When the assessment is complete, the results display as shown in the following graphic:



For single databases or pooled databases in Azure SQL Database, the assessments identify feature parity issues and migration blocking issues for deploying to a single database or pooled database.

- The **SQL Server feature parity** category provides a comprehensive set of recommendations, alternative approaches available in Azure, and mitigating steps to help you plan the effort into your migration projects.
  - The **Compatibility issues** category identifies partially supported or unsupported features that reflect compatibility issues that might block migrating on-premises SQL Server database(s) to Azure SQL Database. Recommendations are also provided to help you address those issues.
6. Review the assessment results for migration blocking issues and feature parity issues by selecting the

specific options.

## Migrate the sample schema

After you're comfortable with the assessment and satisfied that the selected database is a viable candidate for migration to a single database or pooled database in Azure SQL Database, use DMA to migrate the schema to Azure SQL Database.

### NOTE

Before you create a migration project in Data Migration Assistant, be sure that you have already provisioned an Azure SQL database as mentioned in the prerequisites. For purposes of this tutorial, the name of the Azure SQL Database is assumed to be **AdventureWorksAzure**, but you can provide whatever name you wish.

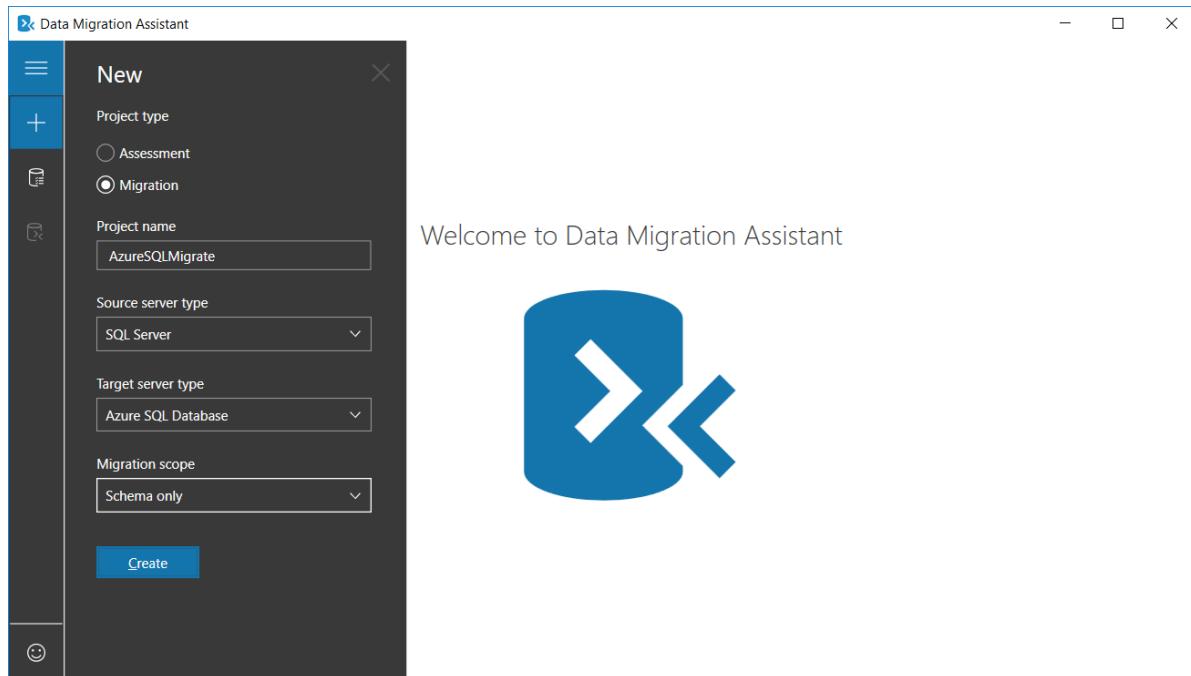
### IMPORTANT

If you use SSIS, DMA does not currently support the migration of source SSISDB, but you can redeploy your SSIS projects/packages to the destination SSISDB hosted by Azure SQL Database. For more information about migrating SSIS packages, see the article [Migrate SQL Server Integration Services packages to Azure](#).

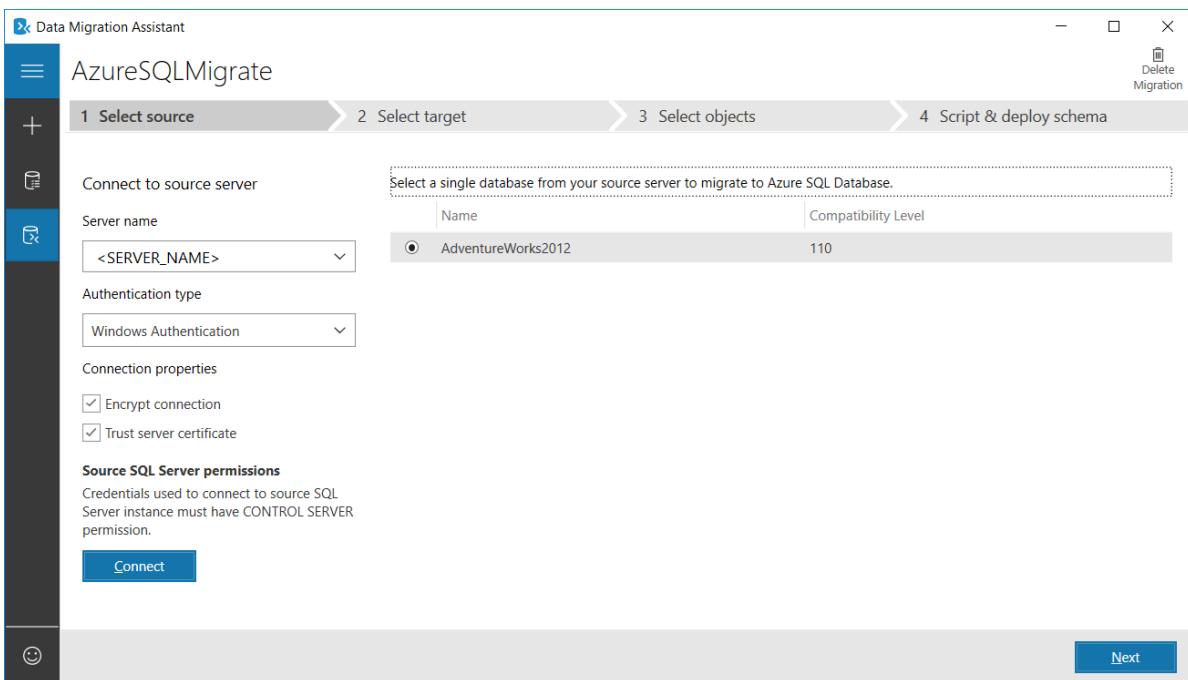
To migrate the **AdventureWorks2012** schema to a single database or pooled database Azure SQL Database, perform the following steps:

1. In the Data Migration Assistant, select the New (+) icon, and then under **Project type**, select **Migration**.
2. Specify a project name, in the **Source server type** text box, select **SQL Server**, and then in the **Target server type** text box, select **Azure SQL Database**.
3. Under **Migration Scope**, select **Schema only**.

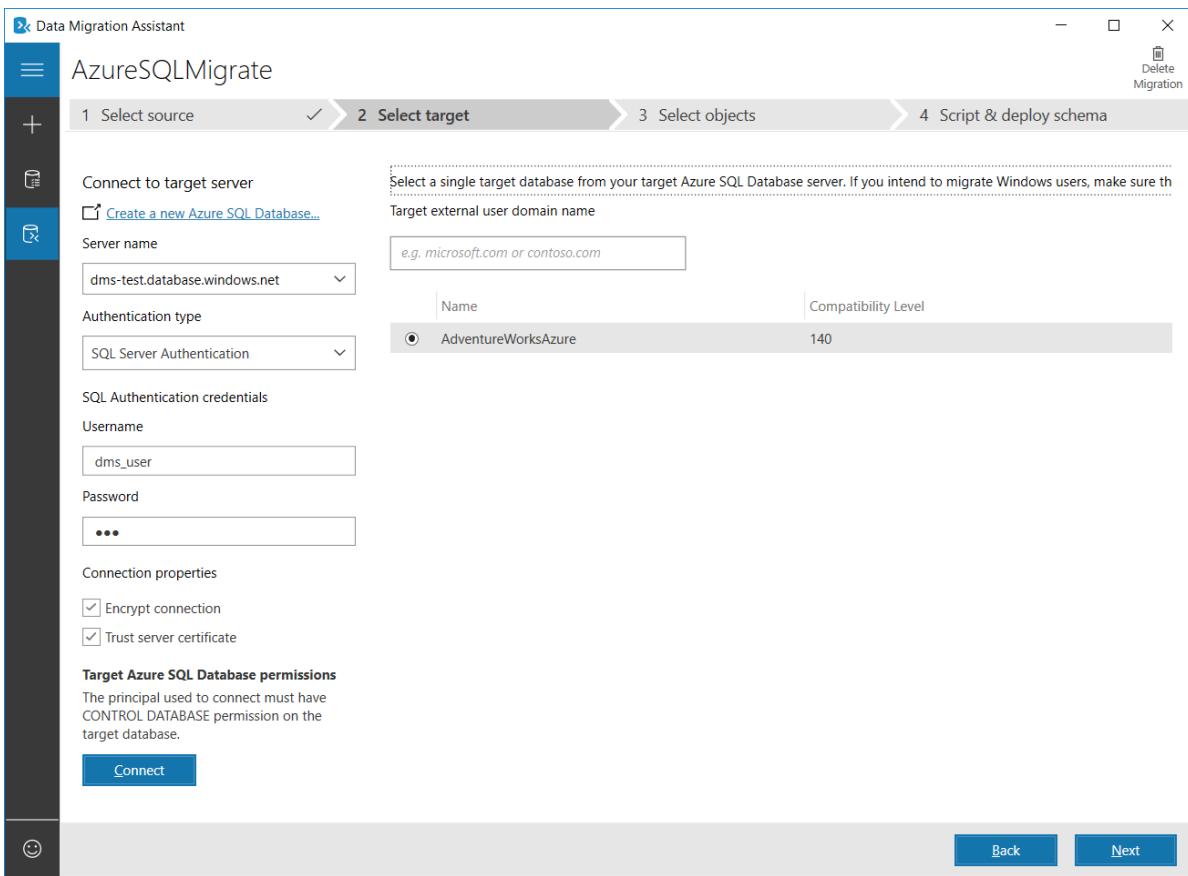
After performing the previous steps, the Data Migration Assistant interface should appear as shown in the following graphic:



4. Select **Create** to create the project.
5. In the Data Migration Assistant, specify the source connection details for your SQL Server, select **Connect**, and then select the **AdventureWorks2012** database.

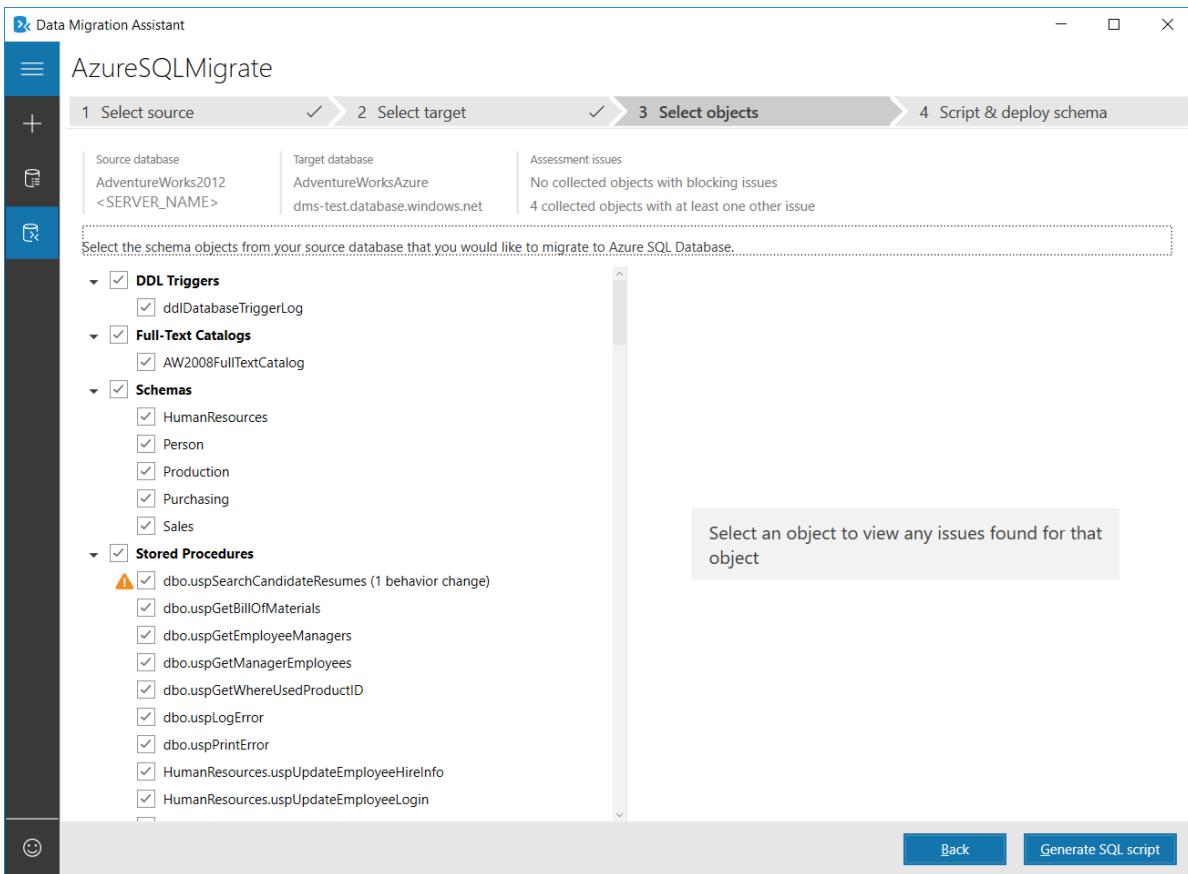


6. Select **Next**, under **Connect to target server**, specify the target connection details for the Azure SQL Database, select **Connect**, and then select the **AdventureWorksAzure** database you had pre-provisioned in Azure SQL Database.

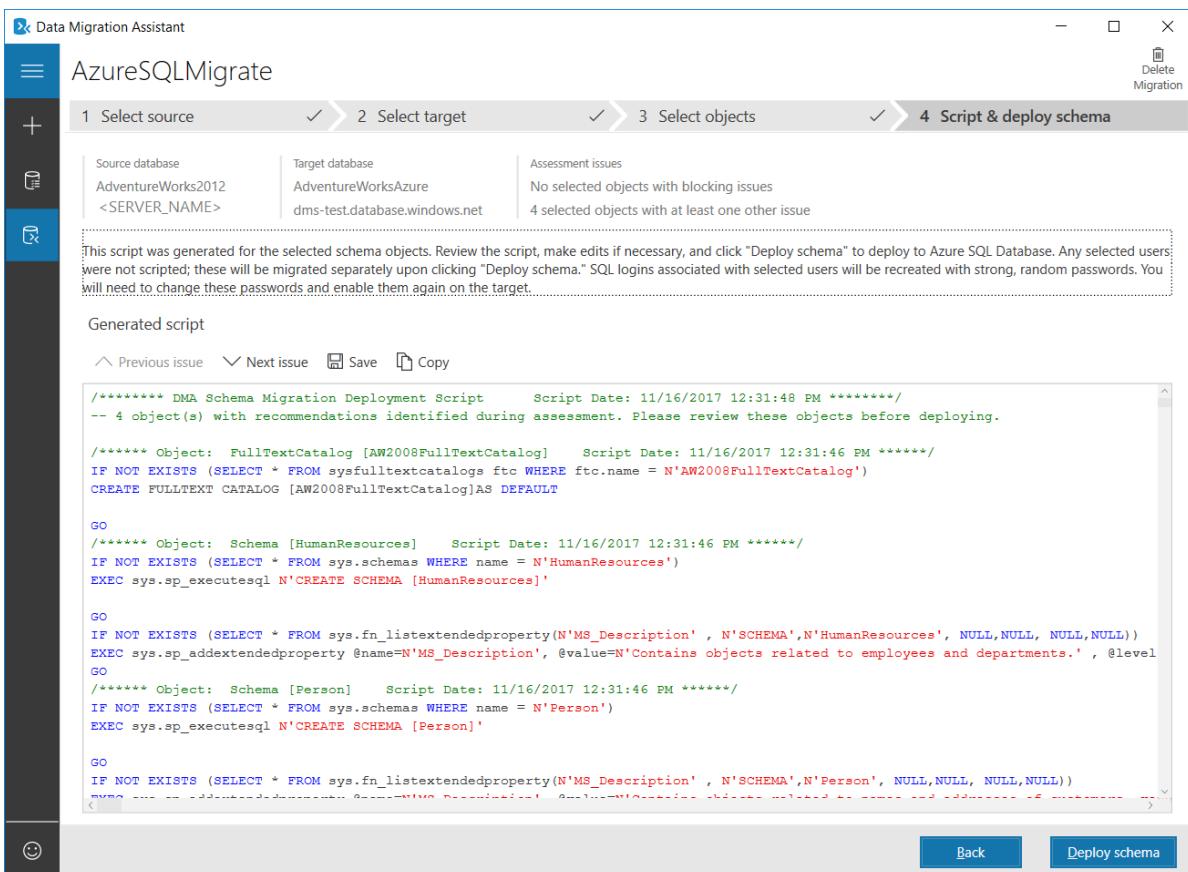


7. Select **Next** to advance to the **Select objects** screen, on which you can specify the schema objects in the **AdventureWorks2012** database that need to be deployed to Azure SQL Database.

By default, all objects are selected.



#### 8. Select **Generate SQL script** to create the SQL scripts, and then review the scripts for any errors.



#### 9. Select **Deploy schema** to deploy the schema to Azure SQL Database, and then after the schema is deployed, check the target server for any anomalies.

This screenshot shows the Microsoft Data Migration Assistant interface for migrating data from an on-premises database to Azure SQL Database. The project name is 'AzureSQLMigrate'. The current step is '4 Script & deploy schema'. The 'Generated script' pane displays the T-SQL commands for schema migration, including creating a FullTextCatalog and schemas. The 'Deployment results' pane shows a list of 2,236 commands executed successfully, with green checkmarks next to each entry.

## Register the Microsoft.DataMigration resource provider

1. Sign in to the Azure portal. Search for and select **Subscriptions**.

This screenshot shows the Microsoft Azure portal's 'Subscriptions' blade. The user has signed in with an account named 'admin@contoso.com'. The 'Subscriptions' link is highlighted with a red box. Other options shown include 'Event Grid Subscriptions', 'Resource groups', and 'Manage subscriptions in the Billing/Account Center'. The right side of the blade displays marketplace items like 'Haivision StreamSRT Standard Annual Subscription' and documentation links such as 'Programmatically create Azure subscriptions - Azure ...' and 'Get billing ownership of Azure subscriptions | Microsoft Docs'.

2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

Home > Subscriptions > Contoso, Ltd Subscription - Resource providers

**Subscriptions**

Contoso

+ Add

Showing subscriptions in Contoso. Don't see a subscription?  
Switch directories

My role (1) Status (1)

8 selected 3 selected

Apply

Show only subscriptions selected in the global subscriptions filter (1)

Search to filter items...

Subscription... ↑ ↓

Contoso - ... 01234567-89ab ... ⋮

Resource providers

Resource groups Resources Usage + quotas Policies Management certificates My permissions Deployments Properties Resource locks

Support + troubleshooting

New support request

3. Search for migration, and then select **Register** for **Microsoft.DataMigration**.

Home > Subscriptions > Contoso, Ltd Subscription - Resource provider

Contoso, Ltd Subscription - Resource provider

Subscription

Search (Ctrl+ /)

Register Unregister Refresh

Migration

Provider	Status
Microsoft.DataMigration	Registering

Resource groups Resources Usage + quotas Policies Management certificates My permissions Deployments Properties Resource locks

Support + troubleshooting

New support request

## Create an instance

1. In the Azure portal menu or on the **Home** page, select **Create a resource**. Search for and select **Azure Database Migration Service**.

The screenshot shows the Azure Marketplace search results. A red box highlights the search bar at the top containing the text 'Azure Database Migration Service'. Below the search bar, there are two tabs: 'Azure Marketplace' and 'See all', followed by a 'Popular' section. The 'Popular' section contains five items: 'Get started' (Windows Server 2016 Datacenter Quickstart tutorial), 'Recently created' (Ubuntu Server 18.04 LTS Learn more), 'AI + Machine Learning' (Analytics), 'Blockchain' (Web App), and 'Web App'.

2. On the **Azure Database Migration Service** screen, select **Create**.

The screenshot shows the 'Create' screen for the Azure Database Migration Service. At the top, it says 'Home > New > Azure Database Migration Service'. The main title is 'Azure Database Migration Service' with a 'Create' button highlighted by a red box. Below the title, there's a Microsoft logo and a 'Save for later' link. The 'Overview' tab is selected. Under 'Common scenarios:', there are three bullet points: 'SQL Server → Azure SQL Database', 'SQL Server → Azure SQL Database Managed Instance', and 'MongoDB → Azure Cosmos DB'.

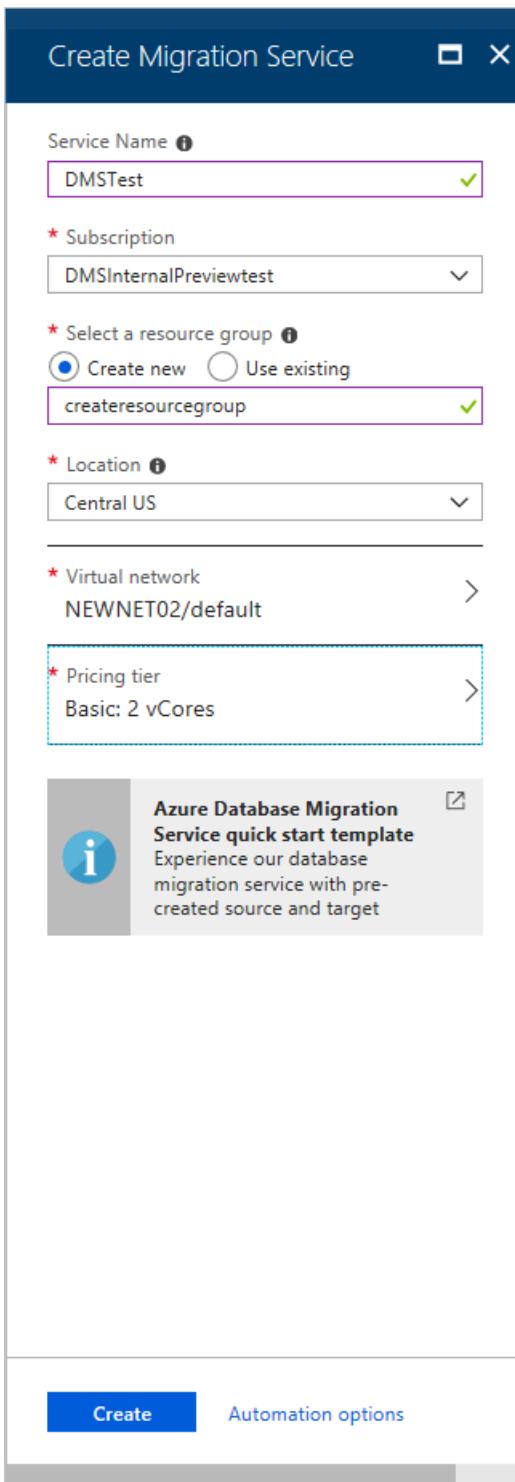
3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.
4. Select the location in which you want to create the instance of Azure Database Migration Service.
5. Select an existing virtual network or create a new one.

The virtual network provides Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Database instance.

For more information about how to create a virtual network in the Azure portal, see the article [Create a virtual network using the Azure portal](#).

6. Select a pricing tier.

For more information on costs and pricing tiers, see the [pricing page](#).

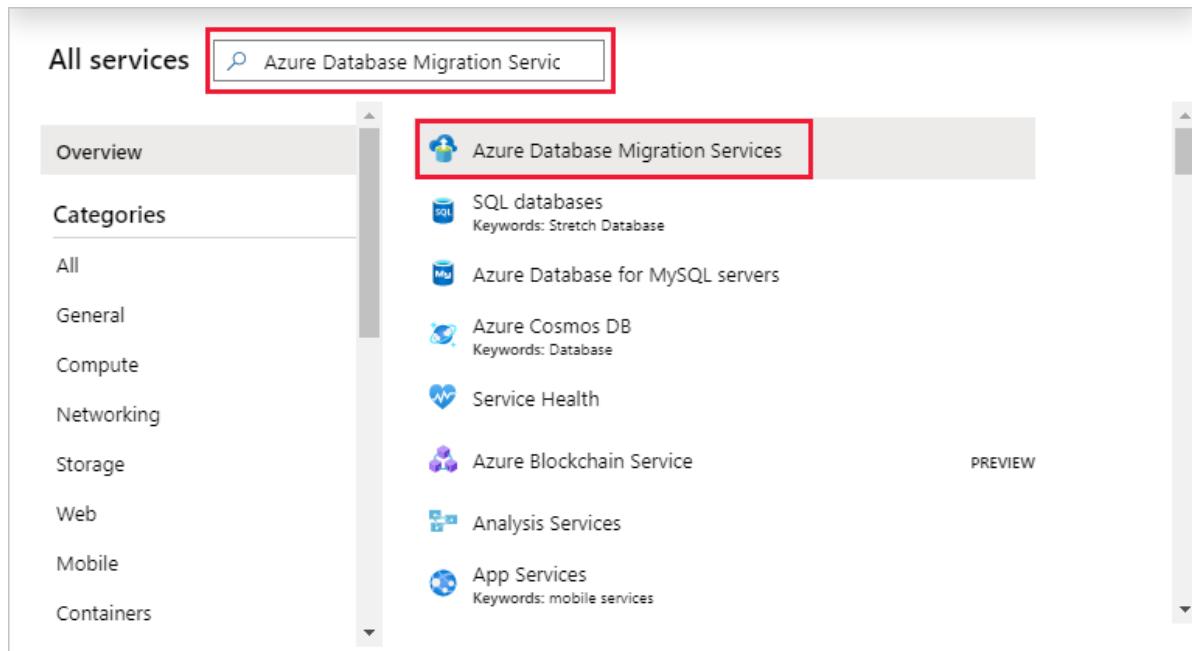


7. Select **Create** to create the service.

## Create a migration project

After the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal menu, select **All services**. Search for and select **Azure Database Migration Services**.

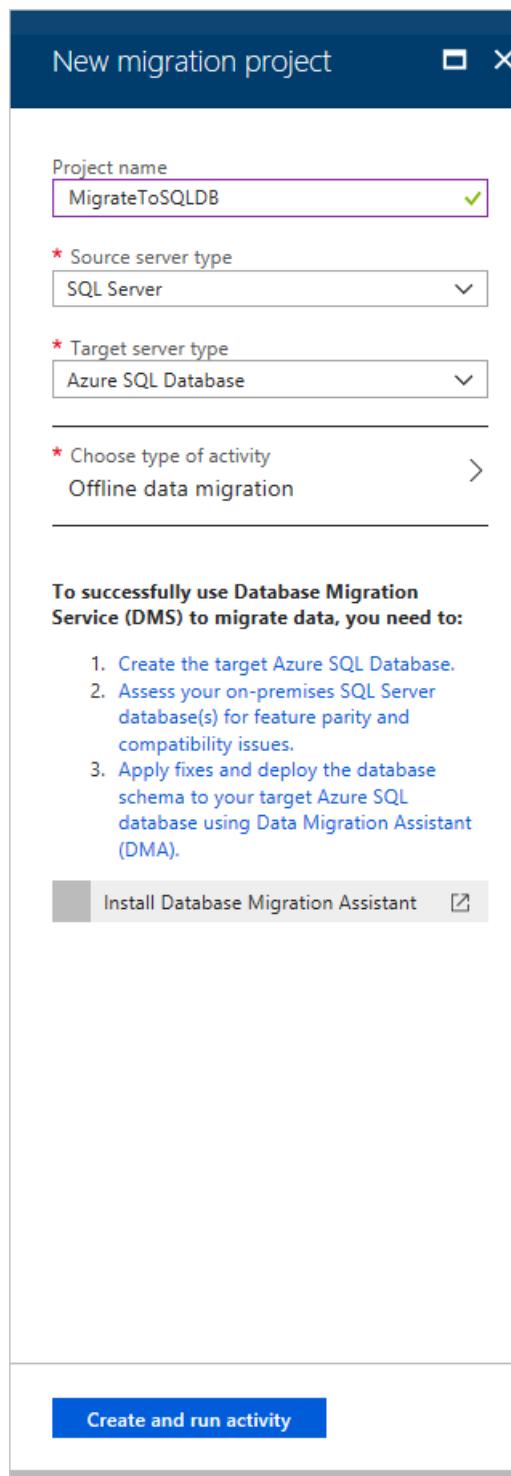


2. On the **Azure Database Migration Services** screen, select the Azure Database Migration Service instance that you created.

3. Select **New Migration Project**.

The screenshot shows the Azure portal's service details page for "DMSTest". A red box highlights the "New Migration Project" button in the top right corner. The left pane shows a list of resources: "DMSTest" (selected), "Activity log", "Access control (IAM)", "Tags", "Settings", "Configuration", "Hybrid", and "Properties". The right pane displays service details: Resource group "createresourcegroup", Status "Online", Virtual network & IP Address "NEWNET02/subnets/default 10.0.0.4", Location "Central US", Subscription "Contoso, Ltd. Subscription", Subsc "12345", SKU "Standard: 1 vCores", Service "4.4.46", and Tags (change) "Click here to add tags".

4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database**, and then for **Choose type of activity**, select **Offline data migration**.



5. Select **Create and run activity** to create the project and run the migration activity.

## Specify source details

1. On the **Migration source detail** screen, specify the connection details for the source SQL Server instance.

Make sure to use a Fully Qualified Domain Name (FQDN) for the source SQL Server instance name. You can also use the IP Address for situations in which DNS name resolution isn't possible.

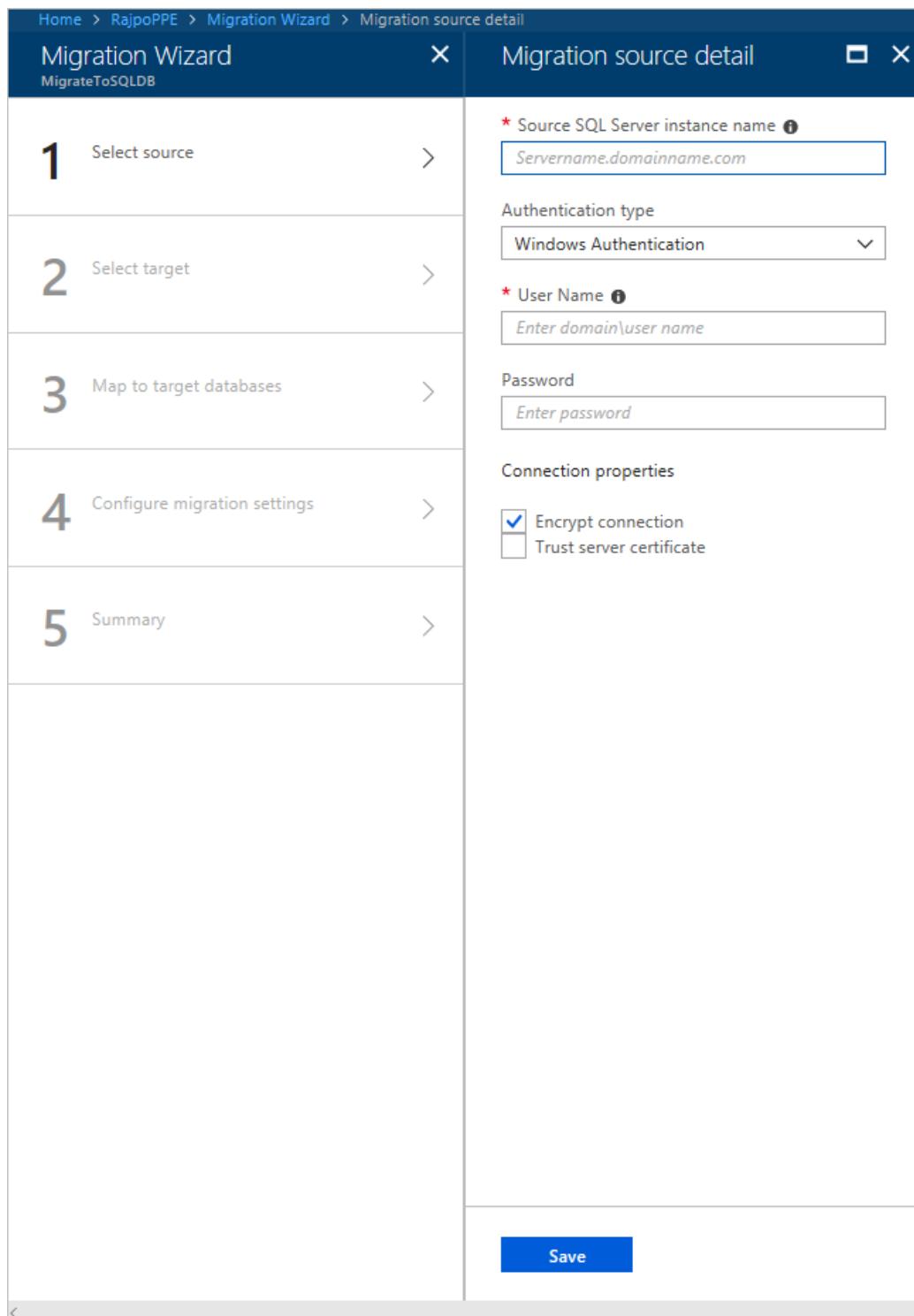
2. If you have not installed a trusted certificate on your source server, select the **Trust server certificate** check box.

When a trusted certificate is not installed, SQL Server generates a self-signed certificate when the instance is started. This certificate is used to encrypt the credentials for client connections.

### Caution

SSL connections that are encrypted using a self-signed certificate do not provide strong security. They are

susceptible to man-in-the-middle attacks. You should not rely on SSL using self-signed certificates in a production environment or on servers that are connected to the internet.

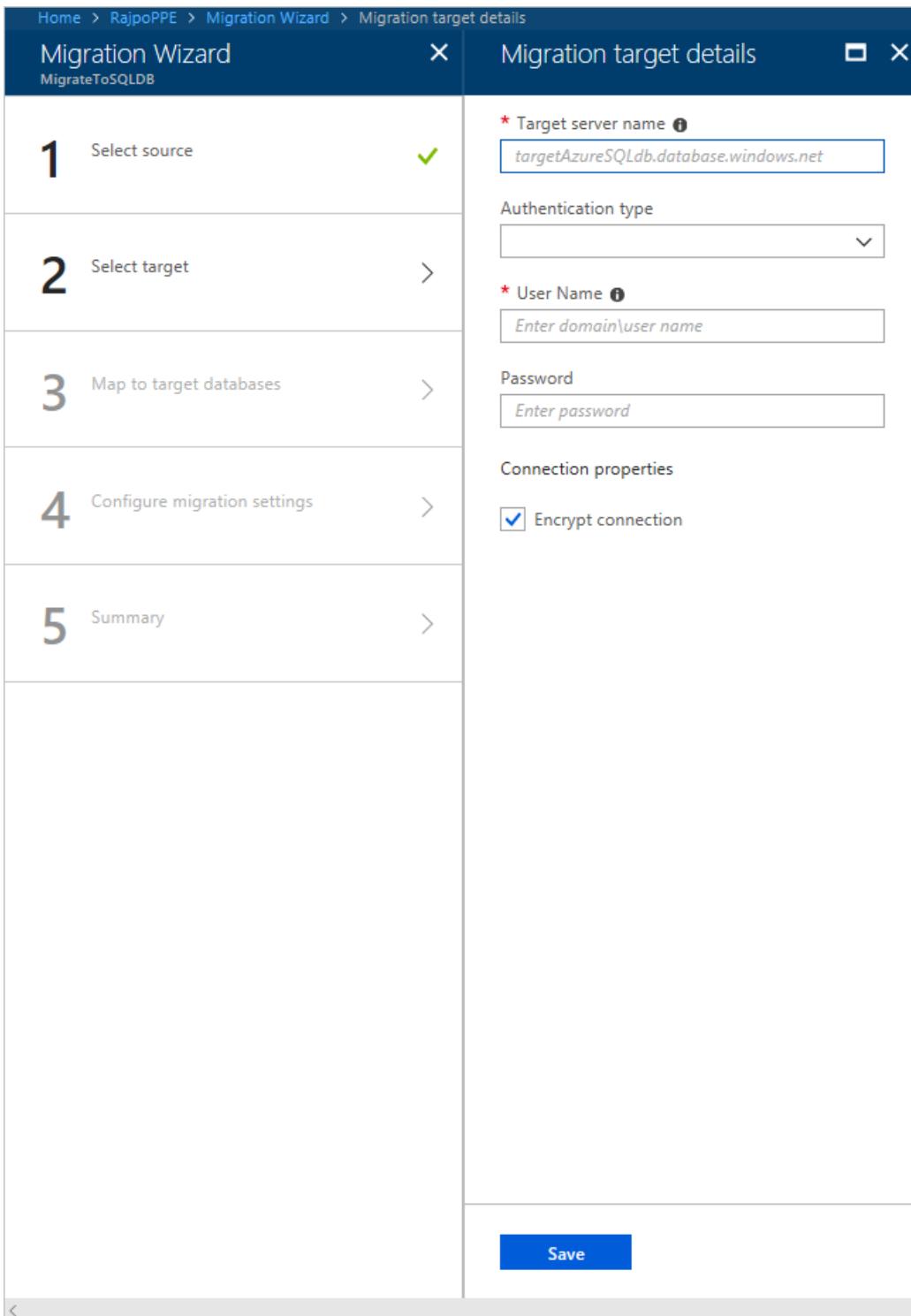


#### IMPORTANT

If you use SSIS, DMS does not currently support the migration of source SSISDB, but you can redeploy your SSIS projects/packages to the destination SSISDB hosted by Azure SQL Database. For more information about migrating SSIS packages, see the article [Migrate SQL Server Integration Services packages to Azure](#).

## Specify target details

1. Select **Save**, and then on the **Migration target details** screen, specify the connection details for the target Azure SQL Database Server, which is the pre-provisioned Azure SQL Database to which the **AdventureWorks2012** schema was deployed by using the Data Migration Assistant.



2. Select **Save**, and then on the **Map to target databases** screen, map the source and the target database for migration.

If the target database contains the same database name as the source database, Azure Database Migration Service selects the target database by default.

**Migration Wizard**  
MigrateToSQLDB

**Map to target databases**

Set the source database to read-only mode during production migrations, to preserve the data consistency and prevent modification of data during the migration. This operation will rollback any active transactions in the source database. The source databases remain in read-only mode after the migration.

SOURCE DATABASE	SIZE	TARGET DATABASE	SET SOURCE DB READ-ONLY
AdventureWorks2012	230.06 MB	AdventureWorksAzure	<input checked="" type="checkbox"/>
contosopayrolldb	5.00 MB		<input type="checkbox"/>
HRMinDowntime	147.31 MB		<input type="checkbox"/>
SitesEE	372.00 MB		<input type="checkbox"/>
StackOverflowEE	486.00 MB		<input type="checkbox"/>
test	21.00 MB		<input type="checkbox"/>

**Save**

3. Select **Save**, on the **Select tables** screen, expand the table listing, and then review the list of affected fields.

Azure Database Migration Service auto selects all the empty source tables that exist on the target Azure SQL Database instance. If you want to remigrate tables that already include data, you need to explicitly select the tables on this blade.

**Migration Wizard**  
MigrateToSQLDB

**Select tables**

Database settings

AdventureWorks2012 71 of 89

NAME	DESCRIPTION
Person.Address	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Person.AddressType	The target table is not empty. If selected for migration, all the records in the table will be deleted...
dbo.AWBuildVersion	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Production.BillOfMat...	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Person.BusinessEntity	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Person.BusinessEntit...	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Person.BusinessEntit...	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Person.ContactType	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Person.CountryRegion	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Sales.CountryRegion...	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Sales.CreditCard	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Production.Culture	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Sales.Currency	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Sales.CurrencyRate	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Sales.Customer	The target table is not empty. If selected for migration, all the records in the table will be deleted...
dbo.DatabaseLog	The target table is not empty. If selected for migration, all the records in the table will be deleted...
HumanResources.De...	The target table is not empty. If selected for migration, all the records in the table will be deleted...
Production.Document	

**Save**

- Select **Save**, on the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity.
- Expand the **Validation option** section to display the **Choose validation option** screen, and then specify whether to validate the migrated databases for **Schema comparison**, **Data consistency**, and **Query correctness**.

The screenshot shows three overlapping windows:

- Migration Wizard**: Step 5, Summary. It lists steps 1-4 as completed (green checkmark) and step 5 as in progress (>). It shows source and target server details, the database to migrate (1 of 6), and the type of activity (Offline data migration).
- Migration summary**: A modal window with a note about upgrading Azure Database tiers if DTU purchase model is used. It also displays the migration project name (MigrateToSQLDB) and the activity name (testmigration, highlighted in purple).
- Choose validation option**: A modal window where the "Validate my database(s)" radio button is selected. It includes a list of validation options: Schema comparison (selected), Data consistency, and Query correctness.

- Select **Save**, review the summary to ensure that the source and target details match what you previously specified.

**Migration Wizard**

MigrateToSQLDB

**Migration summary**

Step	Description	Status
1	Select source	✓
2	Select target	✓
3	Map to target databases	✓
4	Configure migration settings	✓
5	Summary	>

Please consider upgrading your Azure SQL Database to Premium tier (for example P11 or P15 for singleton databases) if DTU purchase model is used or Business critical tier (for example 8 or 16 vCORE) in the case of vCORE model. You can revert to a lower tier once migration is complete.

Upgrade Your Database Service Tiers

**Migration project name**  
MigrateToSQLDB

**Activity name**  
testmigration ✓

**Source server name**  
10.105.129.164

**Source server version**  
SQL Server 2012  
11.0.7462.6

**Target server name**  
dmazuresqlserver.database.windows.net

**Target server version**  
Azure SQL Database  
12.0.2000.8

**Database(s) to migrate**  
1 of 6

**Type of activity**  
Offline data migration

---

\* Validation option >  
3/3 options selected

---

**Run migration**

## Run the migration

- Select **Run migration**.

The migration activity window appears, and the **Status** of the activity is **Pending**.

The screenshot shows the 'testmigration' migration activity screen. At the top, there are four buttons: 'Delete migration', 'Stop migration', 'Refresh', and 'Download report'. Below these are sections for 'Source server' (set to '<source server>'), 'Target server' (set to '<target server>'), 'Source version' (SQL Server 2012), and 'Target version' (Azure SQL Database). A 'Databases' section indicates 1 database. A search bar says 'Search to filter items...'. A table lists the database details:

NAME	STATUS	SIZE	MIGRATION ...	DURATION
AdventureWork...	Pending			

## Monitor the migration

1. On the migration activity screen, select **Refresh** to update the display until the **Status** of the migration shows as **Completed**.

The screenshot shows the 'testmigration' migration activity screen after the migration has completed. The 'Status' column for the database row now shows 'Completed'. The rest of the interface is identical to the previous screenshot.

2. After the migration completes, select **Download report** to get a report listing the details associated with the migration process.
3. Verify the target database(s) on the target Azure SQL Database server.

### Additional resources

- [SQL migration using Azure Data Migration Service](#) hands-on lab.
- For information about known issues and limitations when performing online migrations to Azure SQL Database, see the article [Known issues and workarounds with Azure SQL Database online migrations](#).
- For information about Azure Database Migration Service, see the article [What is Azure Database Migration Service?](#).
- For information about Azure SQL Database, see the article [What is the Azure SQL Database service?](#).

# Tutorial: Add an Azure SQL Database single database to a failover group

12/30/2019 • 21 minutes to read • [Edit Online](#)

Configure a failover group for an Azure SQL Database single database and test failover using either the Azure portal, PowerShell, or Azure CLI. In this tutorial, you will learn how to:

- Create an Azure SQL Database single database.
- Create a [failover group](#) for a single database between two logical SQL servers.
- Test failover.

## Prerequisites

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To complete this tutorial, make sure you have:

- An Azure subscription. [Create a free account](#) if you don't already have one.

## 1 - Create a single database

In this step, you will create an Azure SQL Database single database.

### IMPORTANT

Be sure to set up firewall rules to use the public IP address of the computer you're using to complete this article.

For information, see [Create a database-level firewall rule](#) or to determine the IP address used for the server-level firewall rule for your computer see [Create a server-level firewall](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Create your resource group and single database using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type *Azure SQL* in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select **+ Add** to open the **Select SQL deployment option** page. You can view additional information about the different databases by selecting **Show details** on the **Databases** tile.
3. Select **Create**:

The screenshot shows the 'Select SQL deployment option' page. It features three main sections: 'SQL databases' (best for modern cloud applications), 'SQL managed instances' (best for most migrations to the cloud), and 'SQL virtual machines' (best for migrations and applications requiring OS-level access). Each section includes a 'Create' button and a 'Show details' link.

4. On the **Basics** tab, in the **Project Details** section, type or select the following values:

- **Subscription:** Drop down and select the correct subscription, if it doesn't appear.
- **Resource group:** Select **Create new**, type `myResourceGroup`, and select **OK**.

The screenshot shows the 'Create SQL Database' Basics tab. It includes a 'Project details' section where users can select a subscription and a resource group. The 'Subscription' dropdown is highlighted with a purple border, and the 'Resource group' dropdown also has a purple border, indicating they are the focus of the step. A note at the top says: 'Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize.' A 'Learn more' link is also present.

5. In the **Database Details** section, type or select the following values:

- **Database name:** Enter `mySampleDatabase`.
- **Server:** Select **Create new**, enter the following values and then select **Select**.
  - **Server name:** Type `mysqlserver`; along with some numbers for uniqueness.
  - **Server admin login:** Type `azureuser`.
  - **Password:** Type a complex password that meets password requirements.
  - **Location:** Choose a location from the drop-down, such as `West US`.

New server

\* Server name  
mysqlserver .database.windows.net

\* Server admin login  
azureuser

\* Password  
.....

\* Confirm password  
.....

\* Location  
West US

Allow Azure services to access server

**IMPORTANT**

Remember to record the server admin login and password so you can log in to the server and databases for this and other quickstarts. If you forget your login or password, you can get the login name or reset the password on the **SQL server** page. To open the **SQL server** page, select the server name on the database **Overview** page after database creation.

- **Want to use SQL elastic pool:** Select the **No** option.
- **Compute + storage:** Select **Configure database**.

DATABASE DETAILS

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

\* Database name  
mySampleDatabase

\* Server [\(new\) mysqlserver \( West US \)](#)  
[Create new](#)

\* Want to use SQL elastic pool? [?](#)  Yes  No

\* Compute + storage [?](#)  
**Standard S0**  
10 DTUs, 1 GB storage  
[Configure database](#)

\* Resource group [?](#)  
(New) myResourceGroup

- Select **Provisioned**.

**Compute tier**

- Provisioned** (selected) 
 Compute resources are pre-allocated  
Billed per hour based on vCores configured
- Serverless** 
 Compute resources are auto-scaled  
Billed per second based on vCores used

**Compute Hardware**  
Select a hardware generation best suited to your application. Select from General purpose, compute-optimized and memory optimized compute.

**Hardware Configuration** Gen5  
up to 80 vCores, up to 408 GB memory [Change configuration](#)

**Save money**  
Save up to 55% with a license you already own. Already have a SQL Server license? [?](#)

Yes  No

**vCores** [How do vCores compare with DTUs?](#) [?](#)

2 vCores

**Data max size** [?](#)

32 GB 1 TB 32 GB **9.6 GB LOG SPACE ALLOCATED**

**Cost summary**

**Gen5 - General Purpose (GP\_Gen5\_2)**  
Cost per vCore (in USD)  
**vCores selected** x 2

Cost per GB (in USD)  
**Max storage selected (in GB)** x

**ESTIMATED COST / MONTH**

**Apply**

- Review the settings for **vCores**, and **Data max size**. Change these as desired.
  - Optionally, you can also select **Change configuration** to change the hardware generation.
- Select **Apply**.

## 6. Select the **Networking** tab and decide if you want to [Allow Azure services and resources to access this server](#), or add a [private endpoint](#).

**Create SQL Database** Microsoft

**Basics** **Networking** [Additional settings](#) [Tags](#) [Review + create](#)

Configure network access and connectivity for your server. The configuration selected below will apply to the selected server 'carlrbperfsrv' and all databases it manages. [Learn more](#) [?](#)

**Firewall rules**

The settings displayed below are read-only. They can be modified from the "Firewalls and virtual networks" blade after database creation. [Learn more](#) [?](#)

Allow Azure services and resources to access this server  Yes  No

**Private endpoints (preview)**

Private endpoint connections are associated with a private IP address within a Virtual Network. The list below shows all the private endpoint connections for this server. Note that private endpoint connections are defined at the server level and they provide access to all databases in the server. [Learn more](#) [?](#)

[+ Add private endpoint](#)

Name	Subscription
<i>Click on add to create private endpoint</i>	

7. Select the **Additional settings** tab.
8. In the **Data source** section, under **Use existing data**, select **Sample**.

Home > Azure SQL > Select SQL deployment option > Create SQL Database

## Create SQL Database

Microsoft

Basics Networking Additional settings Tags Review + create

Customize additional configuration parameters including collation & sample data.

**Data source**

Start with a blank database, restore from a backup or select sample data to populate your new database.

Use existing data \*

None Backup Sample

AdventureWorksLT will be created as the sample database.

**Database collation**

Database collation defines the rules that sort and compare data, and cannot be changed after database creation. The default database collation is SQL\_Latin1\_General\_CI\_AS. [Learn more](#)

Collation ⓘ SQL\_Latin1\_General\_CI\_AS

**Advanced data security**

Protect your data using advanced data security, a unified security package including data classification, vulnerability assessment and advanced threat protection for your server. [Learn more](#)

**Advanced data security has already been enabled on the selected server.**

### IMPORTANT

Make sure to select the **Sample (AdventureWorksLT)** data so you can follow easily this and other Azure SQL Database quickstarts that use this data.

9. Leave the rest of the values as default and select **Review + Create** at the bottom of the form.
10. Review the final settings and select **Create**.
11. On the **SQL Database** form, select **Create** to deploy and provision the resource group, server, and database.

## 2 - Create the failover group

In this step, you will create a [failover group](#) between an existing Azure SQL server and a new Azure SQL server in another region. Then add the sample database to the failover group.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Create your failover group and add your single database to it using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select the single database created in section 1, such as `mySampleDatabase`.
3. Select the name of the server under **Server name** to open the settings for the server.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase)

**mySampleDatabase (mysqlserver/mySampleDatabase)**

SQL database

Search (Ctrl+ /) Copy Restore Export Set server firewall Delete Connect with... Feedback

Resource group (change) : myResourceGroup Server name : **mysqlserver.database.windows.net**

Status : Online Elastic pool : No elastic pool

Location : East US Connection strings : Show database connection strings

Overview Activity log

4. Select **Failover groups** under the **Settings** pane, and then select **Add group** to create a new failover group.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase) > mysqlserver- Failover groups

**mysqlserver - Failover groups**

SQL server

Search (Ctrl+ /) + Add group Refresh

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings Quick start Failover groups

Failover group are a SQL server feature designed to automatically manage repl

NAME	PRIMARY SERVER	SECOND
You have no group created		

5. On the **Failover Group** page, enter or select the following values, and then select **Create**:

- **Failover group name:** Type in a unique failover group name, such as `failovergrouptutorial`.
- **Secondary server:** Select the option to *configure required settings* and then choose to **Create a new server**. Alternatively, you can choose an already-existing server as the secondary server. After entering the following values, select **Select**.
  - **Server name:** Type in a unique name for the secondary server, such as `mysqlsecondary`.
  - **Server admin login:** Type `azureuser`
  - **Password:** Type a complex password that meets password requirements.
  - **Location:** Choose a location from the drop-down, such as `East US`. This location cannot be the same location as your primary server.

#### NOTE

The server login and firewall settings must match that of your primary server.

The screenshot shows three overlapping windows:

- Failover group**: Shows a failover group named "failovergrouptutorial" with ".database.windows.net". A red box highlights the "Secondary server" section.
- Server**: Shows a "Create a new server" button with a red box around it.
- New server**: Shows fields for "Server name" (mysqlsecondary), "Server admin login" (azureuser), "Password", "Confirm password", "Location" (East US), and a checked "Allow Azure services to access server" checkbox.

- **Databases within the group:** Once a secondary server is selected, this option becomes unlocked. Select it to **Select databases to add** and then choose the database you created in section 1. Adding the database to the failover group will automatically start the geo-replication process.

The screenshot shows two windows:

- Failover group**: Same configuration as the previous screenshot.
- Databases**: Shows a table with one row:
 

	NAME	ROLE	SECONDARY SERVER	STATUS
	mySampleDatabase			Online

 A red box highlights the "Select databases to add" button at the bottom of the left sidebar.

## 3 - Test failover

In this step, you will fail your failover group over to the secondary server, and then fail back using the Azure portal.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Test failover using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select the single database created in the section 2, such as `mySampleDatabase`.
3. Select the name of the server under **Server name** to open the settings for the server.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase)

**mySampleDatabase (mysqlserver/mySampleDatabase)**

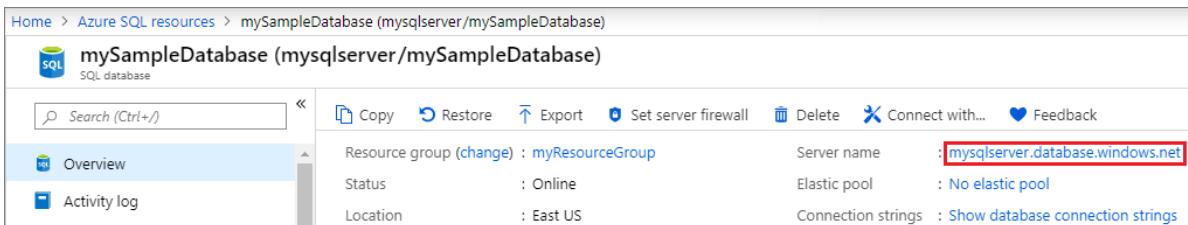
SQL database

Search (Ctrl+ /) Copy Restore Export Set server firewall Delete Connect with... Feedback

Overview Resource group (change) : myResourceGroup Server name : mysqlserver.database.windows.net

Activity log Status : Online Elastic pool : No elastic pool

Location : East US Connection strings : Show database connection strings



4. Select **Failover groups** under the **Settings** pane and then choose the failover group you created in section 2.

Home > Azure SQL resources > mySampleDatabase (mysqlserver/mySampleDatabase) > mysqlserver - Failover groups

**mysqlserver - Failover groups**

SQL server

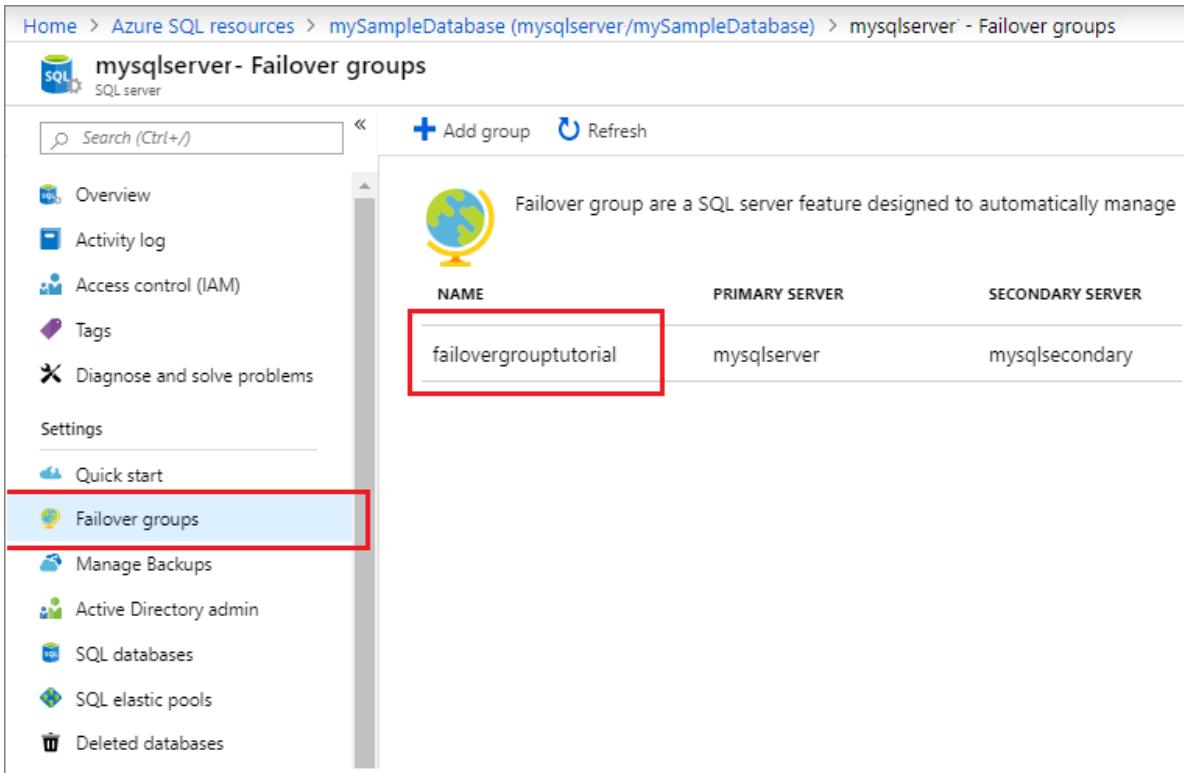
Search (Ctrl+ /) Add group Refresh

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Quick start Failover groups Manage Backups Active Directory admin SQL databases SQL elastic pools Deleted databases

NAME PRIMARY SERVER SECONDARY SERVER

NAME	PRIMARY SERVER	SECONDARY SERVER
failovergrouptutorial	mysqlserver	mysqlsecondary



5. Review which server is primary and which server is secondary.
6. Select **Failover** from the task pane to failover your failover group containing your sample single database.
7. Select **Yes** on the warning that notifies you that TDS sessions will be disconnected.

SERVER	ROLE	READ/WRITE FAILOVER POLICY
<input checked="" type="checkbox"/> mysqlserver (West US)	Primary	Automatic
<input checked="" type="checkbox"/> mysqlsecondary (East US)	Secondary	

8. Review which server is now primary and which server is secondary. If fail over succeeded, the two servers should have swapped roles.
9. Select **Failover** again to fail the servers back to their originally roles.

## Clean up resources

Clean up resources by deleting the resource group.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Delete the resource group using the Azure portal.

1. Navigate to your resource group in the [Azure portal](#).
2. Select **Delete resource group** to delete all the resources in the group, as well as the resource group itself.
3. Type the name of the resource group, `myResourceGroup`, in the textbox, and then select **Delete** to delete the resource group.

### IMPORTANT

If you want to keep the resource group but delete the secondary database, remove it from the failover group before deleting it. Deleting a secondary database before it is removed from the failover group can cause unpredictable behavior.

## Full scripts

- [PowerShell](#)
- [Azure CLI](#)

- [Portal](#)

There are no scripts available for the Azure portal.

You can find other Azure SQL Database scripts here: [Azure PowerShell](#) and [Azure CLI](#).

## Next steps

In this tutorial, you added an Azure SQL Database single database to a failover group, and tested failover. You learned how to:

- Create an Azure SQL Database single database.
- Create a [failover group](#) for a single database between two logical SQL servers.
- Test failover.

Advance to the next tutorial on how to add your elastic pool to a failover group.

[Tutorial: Add an Azure SQL Database elastic pool to a failover group](#)

# Tutorial: Implement a geo-distributed database

11/22/2019 • 7 minutes to read • [Edit Online](#)

Configure an Azure SQL database and application for failover to a remote region and test a failover plan. You learn how to:

- Create a [failover group](#)
- Run a Java application to query an Azure SQL database
- Test failover

If you don't have an Azure subscription, [create a free account](#) before you begin.

## Prerequisites

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

To complete the tutorial, make sure you've installed the following items:

- [Azure PowerShell](#)
- A single database in Azure SQL Database. To create one use,
  - [Portal](#)
  - [CLI](#)
  - [PowerShell](#)

### NOTE

The tutorial uses the *AdventureWorksLT* sample database.

- Java and Maven, see [Build an app using SQL Server](#), highlight **Java** and select your environment, then follow the steps.

## IMPORTANT

Be sure to set up firewall rules to use the public IP address of the computer on which you're performing the steps in this tutorial. Database-level firewall rules will replicate automatically to the secondary server.

For information see [Create a database-level firewall rule](#) or to determine the IP address used for the server-level firewall rule for your computer see [Create a server-level firewall](#).

## Create a failover group

Using Azure PowerShell, create [failover groups](#) between an existing Azure SQL server and a new Azure SQL server in another region. Then add the sample database to the failover group.

- [PowerShell](#)
- [Azure CLI](#)

## IMPORTANT

This sample requires Azure PowerShell Az 1.0 or later. Run `Get-Module -ListAvailable Az` to see which versions are installed. If you need to install, see [Install Azure PowerShell module](#).

Run [Connect-AzAccount](#) to sign in to Azure.

To create a failover group, run the following script:

```
$admin = "<adminName>"
$password = "<password>"
$resourceGroup = "<resourceGroupName>"
$location = "<resourceGroupLocation>"
$server = "<serverName>"
$database = "<databaseName>"
$drLocation = "<disasterRecoveryLocation>"
$drServer = "<disasterRecoveryServerName>"
$failoverGroup = "<globallyUniqueFailoverGroupName>"

create a backup server in the failover region
New-AzSqlServer -ResourceGroupName $resourceGroup -ServerName $drServer `
 -Location $drLocation -SqlAdministratorCredentials $(New-Object -TypeName
System.Management.Automation.PSCredential `
 -ArgumentList $admin, $(ConvertTo-SecureString -String $password -AsPlainText -Force))

create a failover group between the servers
New-AzSqlDatabaseFailoverGroup -ResourceGroupName $resourceGroup -ServerName $server `
 -PartnerServerName $drServer -FailoverGroupName $failoverGroup -FailoverPolicy Automatic -
 GracePeriodWithDataLossHours 2

add the database to the failover group
Get-AzSqlDatabase -ResourceGroupName $resourceGroup -ServerName $server -DatabaseName $database | `
 Add-AzSqlDatabaseToFailoverGroup -ResourceGroupName $resourceGroup -ServerName $server -FailoverGroupName
 $failoverGroup
```

Geo-replication settings can also be changed in the Azure portal, by selecting your database, then **Settings** > **Geo-Replication**.



	SERVER/DATABASE	FAILOVER POLICY	STATUS
<strong>PRIMARY</strong>			
	West US      <server>/<database>	<failovergroup> (Automatic, 2 hours)	Online
<strong>SECONDARIES</strong>			
	West Europe      <server>/<database>		Readable

## Run the sample project

1. In the console, create a Maven project with the following command:

```
mvn archetype:generate "-DgroupId=com.sql dbsamples" "-DartifactId=SqlDbSample" "-DarchetypeArtifactId=maven-archetype-quickstart" "-Dversion=1.0.0"
```

2. Type **Y** and press **Enter**.

3. Change directories to the new project.

```
cd SqlDbSample
```

4. Using your favorite editor, open the `pom.xml` file in your project folder.

5. Add the Microsoft JDBC Driver for SQL Server dependency by adding the following `dependency` section. The dependency must be pasted within the larger `dependencies` section.

```
<dependency>
<groupId>com.microsoft.sqlserver</groupId>
<artifactId>mssql-jdbc</artifactId>
<version>6.1.0.jre8</version>
</dependency>
```

6. Specify the Java version by adding the `properties` section after the `dependencies` section:

```
<properties>
 <maven.compiler.source>1.8</maven.compiler.source>
 <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

7. Support manifest files by adding the `build` section after the `properties` section:

```
<build>
 <plugins>
 <plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-jar-plugin</artifactId>
 <version>3.0.0</version>
 <configuration>
 <archive>
 <manifest>
 <mainClass>com.sql dbsamples.App</mainClass>
 </manifest>
 </archive>
 </configuration>
 </plugin>
 </plugins>
</build>
```

8. Save and close the `pom.xml` file.

9. Open the `App.java` file located in `..\SqlDbSample\src\main\java\com\sql dbsamples` and replace the contents with the following code:

```
package com.sql dbsamples;

import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Timestamp;
import java.sql.DriverManager;
import java.util.Date;
import java.util.concurrent.TimeUnit;

public class App {

 private static final String FAILOVER_GROUP_NAME = "<your failover group name>"; // add failover
 group name

 private static final String DB_NAME = "<your database>"; // add database name
 private static final String USER = "<your admin>"; // add database user
 private static final String PASSWORD = "<your password>"; // add database password

 private static final String READ_WRITE_URL = String.format("jdbc:" +
 "sqlserver://%.database.windows.net:1433;database=%s;user=%s;password=%s;encrypt=true;" +
 "hostNameInCertificate=*.database.windows.net;loginTimeout=30;", +
 FAILOVER_GROUP_NAME, DB_NAME, USER, PASSWORD);
 private static final String READ_ONLY_URL = String.format("jdbc:" +
 "sqlserver://%.secondary.database.windows.net:1433;database=%s;user=%s;password=%s;encrypt=true;" +
 "hostNameInCertificate=*.database.windows.net;loginTimeout=30;", +
 FAILOVER_GROUP_NAME, DB_NAME, USER, PASSWORD);

 public static void main(String[] args) {
 System.out.println("#####");
 System.out.println("## GEO DISTRIBUTED DATABASE TUTORIAL ##");
 System.out.println("#####");
 System.out.println("");
 }
}
```

```

int highWaterMark = getHighWaterMarkId();

try {
 for(int i = 1; i < 1000; i++) {
 // loop will run for about 1 hour
 System.out.print(i + ": insert on primary " +
 (insertData((highWaterMark + i))?"successful":"failed"));
 TimeUnit.SECONDS.sleep(1);
 System.out.print(", read from secondary " +
 (selectData((highWaterMark + i))?"successful":"failed") + "\n");
 TimeUnit.SECONDS.sleep(3);
 }
} catch(Exception e) {
 e.printStackTrace();
}

private static boolean insertData(int id) {
 // Insert data into the product table with a unique product name so we can find the product again
 String sql = "INSERT INTO SalesLT.Product " +
 "(Name, ProductNumber, Color, StandardCost, ListPrice, SellStartDate) VALUES (?, ?, ?, ?, ?, ?);";

 try (Connection connection = DriverManager.getConnection(READ_WRITE_URL);
 PreparedStatement pstmt = connection.prepareStatement(sql)) {
 pstmt.setString(1, "BrandNewProduct" + id);
 pstmt.setInt(2, 200989 + id + 10000);
 pstmt.setString(3, "Blue");
 pstmt.setDouble(4, 75.00);
 pstmt.setDouble(5, 89.99);
 pstmt.setTimestamp(6, new Timestamp(new Date().getTime()));
 return (1 == pstmt.executeUpdate());
 } catch (Exception e) {
 return false;
 }
}

private static boolean selectData(int id) {
 // Query the data previously inserted into the primary database from the geo replicated database
 String sql = "SELECT Name, Color, ListPrice FROM SalesLT.Product WHERE Name = ?";

 try (Connection connection = DriverManager.getConnection(READ_ONLY_URL);
 PreparedStatement pstmt = connection.prepareStatement(sql)) {
 pstmt.setString(1, "BrandNewProduct" + id);
 try (ResultSet resultSet = pstmt.executeQuery()) {
 return resultSet.next();
 }
 } catch (Exception e) {
 return false;
 }
}

private static int getHighWaterMarkId() {
 // Query the high water mark id stored in the table to be able to make unique inserts
 String sql = "SELECT MAX(ProductId) FROM SalesLT.Product";
 int result = 1;
 try (Connection connection = DriverManager.getConnection(READ_WRITE_URL);
 Statement stmt = connection.createStatement();
 ResultSet resultSet = stmt.executeQuery(sql)) {
 if (resultSet.next()) {
 result = resultSet.getInt(1);
 }
 } catch (Exception e) {
 e.printStackTrace();
 }
 return result;
}
}

```

10. Save and close the *App.java* file.

11. In the command console, run the following command:

```
mvn package
```

12. Start the application that will run for about 1 hour until stopped manually, allowing you time to run the failover test.

```
mvn -q -e exec:java "-Dexec.mainClass=com.sql dbsamples.App"
```

```
#####
GEO DISTRIBUTED DATABASE TUTORIAL
#####
```

```
1. insert on primary successful, read from secondary successful
2. insert on primary successful, read from secondary successful
3. insert on primary successful, read from secondary successful
...
...
```

## Test failover

Run the following scripts to simulate a failover and observe the application results. Notice how some inserts and selects will fail during the database migration.

- [PowerShell](#)
- [Azure CLI](#)

You can check the role of the disaster recovery server during the test with the following command:

```
(Get-AzSqlDatabaseFailoverGroup -FailoverGroupName $failoverGroup `
-ResourceGroupName $resourceGroup -ServerName $drServer).ReplicationRole
```

To test a failover:

1. Start a manual failover of the failover group:

```
Switch-AzSqlDatabaseFailoverGroup -ResourceGroupName $myresourcegroupname `
-ServerName $drServer -FailoverGroupName $failoverGroup
```

2. Revert failover group back to the primary server:

```
Switch-AzSqlDatabaseFailoverGroup -ResourceGroupName $resourceGroup `
-ServerName $server -FailoverGroupName $failoverGroup
```

## Next steps

In this tutorial, you configured an Azure SQL database and application for failover to a remote region and tested a failover plan. You learned how to:

- Create a geo-replication failover group
- Run a Java application to query an Azure SQL database

- Test failover

Advance to the next tutorial on how to migrate using DMS.

[Migrate SQL Server to Azure SQL database managed instance using DMS](#)

# Configure active geo-replication for Azure SQL Database in the Azure portal and initiate failover

12/11/2019 • 3 minutes to read • [Edit Online](#)

This article shows you how to configure [active geo-replication for single and pooled databases](#) in Azure SQL Database using the [Azure portal](#) and to initiate failover.

For information about auto-failover groups with single and pooled databases, see [Best practices of using failover groups with single and pooled databases](#). For information about auto-failover groups with Managed Instances, see [Best practices of using failover groups with managed-instances](#).

## Prerequisites

To configure active geo-replication by using the Azure portal, you need the following resource:

- An Azure SQL database: The primary database that you want to replicate to a different geographical region.

### NOTE

When using Azure portal, you can only create a secondary database within the same subscription as the primary. If secondary database is required to be in a different subscription, use [Create Database REST API](#) or [ALTER DATABASE Transact-SQL API](#).

## Add a secondary database

The following steps create a new secondary database in a geo-replication partnership.

To add a secondary database, you must be the subscription owner or co-owner.

The secondary database has the same name as the primary database and has, by default, the same service tier and compute size. The secondary database can be a single database or a pooled database. For more information, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#). After the secondary is created and seeded, data begins replicating from the primary database to the new secondary database.

### NOTE

If the partner database already exists (for example, as a result of terminating a previous geo-replication relationship) the command fails.

1. In the [Azure portal](#), browse to the database that you want to set up for geo-replication.
2. On the SQL database page, select **geo-replication**, and then select the region to create the secondary database. You can select any region other than the region hosting the primary database, but we recommend the [paired region](#).

WideWorldImporters - Geo-Replication

Select a region on the map or from the Target Regions list to create a secondary database.

Search (Ctrl+ /)

Overview

Activity log

Tags

Diagnose and solve problems

SETTINGS

Quick start

Pricing tier (scale DTUs)

**Geo-Replication**

Auditing & Threat detection

Dynamic data masking

Transparent data encryption

Properties

Locks

Automation script

SERVER/DATABASE STATUS

PRIMARY

Region	Server Name	Status
North Central US	sqldbteam/WideWorldImporte...	Online

SECONDARIES

Geo-Replication is not configured

3. Select or configure the server and pricing tier for the secondary database.

Create secondary

Create geo-replicated secondaries to protect against prolonged datacenter [Learn more](#)

Region  
South Central US

Database name  
WideWorldImporters

Pricing tier  
S2 Standard

\* Secondary type  
Readable

\* Target server  
*Configure required settings*

Elastic database pool

Pin to dashboard

OK

4. Optionally, you can add a secondary database to an elastic pool. To create the secondary database in a pool, click **elastic pool** and select a pool on the target server. A pool must already exist on the target server. This

workflow does not create a pool.

5. Click **Create** to add the secondary.
6. The secondary database is created and the seeding process begins.

SERVER/DATABASE		STATUS
PRIMARY		
North Central US	sqlldbteam/WideWorldImport...	Online
SECONDARIES		
South Central US	server101801/WideWorldImp...	Initializing...

7. When the seeding process is complete, the secondary database displays its status.

SERVER/DATABASE		STATUS
PRIMARY		
North Central US	sqlldbteam/WideWorldImport...	Online
SECONDARIES		
South Central US	server101801/WideWorldImp...	Readable

## Initiate a failover

The secondary database can be switched to become the primary.

1. In the [Azure portal](#), browse to the primary database in the geo-replication partnership.
2. On the SQL Database blade, select **All settings > geo-replication**.
3. In the **SECONDARIES** list, select the database you want to become the new primary and click **Failover**.

**Geo-Replication**  
sqlserver421/adventureworks

Select a region on the map or from the Target Regions list to create a secondary database.

**Japan East**  
Secondary database

**Failover**  Stop Replicatio...

**REGION**  
Japan East

**DATABASE NAME**  
adventureworks

**SERVER**  
ssserver421

**PRICING TIER**  
S0 Standard (10 DTUs)

**STATUS**  
Readable

**SERVER/DATABASE** **STATUS**

**PRIMARY**

<input checked="" type="checkbox"/>	Japan West	sqlserver421/adventureworks	Online
-------------------------------------	------------	-----------------------------	--------

**SECONDARIES**

<input checked="" type="checkbox"/>	Japan East	ssserver421/adventureworks	Readable	...
-------------------------------------	------------	----------------------------	----------	-----

4. Click **Yes** to begin the failover.

The command immediately switches the secondary database into the primary role. This process normally should complete within 30 sec or less.

There is a short period during which both databases are unavailable (on the order of 0 to 25 seconds) while the roles are switched. If the primary database has multiple secondary databases, the command automatically reconfigures the other secondaries to connect to the new primary. The entire operation should take less than a minute to complete under normal circumstances.

#### NOTE

This command is designed for quick recovery of the database in case of an outage. It triggers failover without data synchronization (forced failover). If the primary is online and committing transactions when the command is issued some data loss may occur.

## Remove secondary database

This operation permanently terminates the replication to the secondary database, and changes the role of the secondary to a regular read-write database. If the connectivity to the secondary database is broken, the command succeeds but the secondary does not become read-write until after connectivity is restored.

1. In the [Azure portal](#), browse to the primary database in the geo-replication partnership.

2. On the SQL database page, select **geo-replication**.
3. In the **SECONDARIES** list, select the database you want to remove from the geo-replication partnership.
4. Click **Stop Replication**.

**Geo-Replication**

sqldbteam/WideWorldImporters

Select a region on the map or from the Target Regions list to create a secondary database.

**South Central US**

Secondary database

Failover  Stop Replication

REGION	South Central US
DATABASE NAME	WideWorldImporters
SERVER	server101801
PRICING TIER	S2 Standard (50 DTUs)
STATUS	Readable

**PRIMARY**

<input checked="" type="checkbox"/>	North Central US	sqldbteam/WideWorldImporters...	Online
-------------------------------------	------------------	---------------------------------	--------

**SECONDARIES**

<input checked="" type="checkbox"/>	South Central US	server101801/WideWorldImp...	Readable	...
-------------------------------------	------------------	------------------------------	----------	-----

5. A confirmation window opens. Click **Yes** to remove the database from the geo-replication partnership. (Set it to a read-write database not part of any replication.)

## Next steps

- To learn more about active geo-replication, see [active geo-replication](#).
- To learn about auto-failover groups, see [Auto-failover groups](#)
- For a business continuity overview and scenarios, see [Business continuity overview](#).

# Tutorial: Set up SQL Data Sync between Azure SQL Database and SQL Server on-premises

12/16/2019 • 9 minutes to read • [Edit Online](#)

In this tutorial, you learn how to set up Azure SQL Data Sync by creating a sync group that contains both Azure SQL Database and SQL Server instances. The sync group is custom configured and synchronizes on the schedule you set.

The tutorial assumes you have at least some prior experience with SQL Database and SQL Server.

For an overview of SQL Data Sync, see [Sync data across cloud and on-premises databases with Azure SQL Data Sync](#).

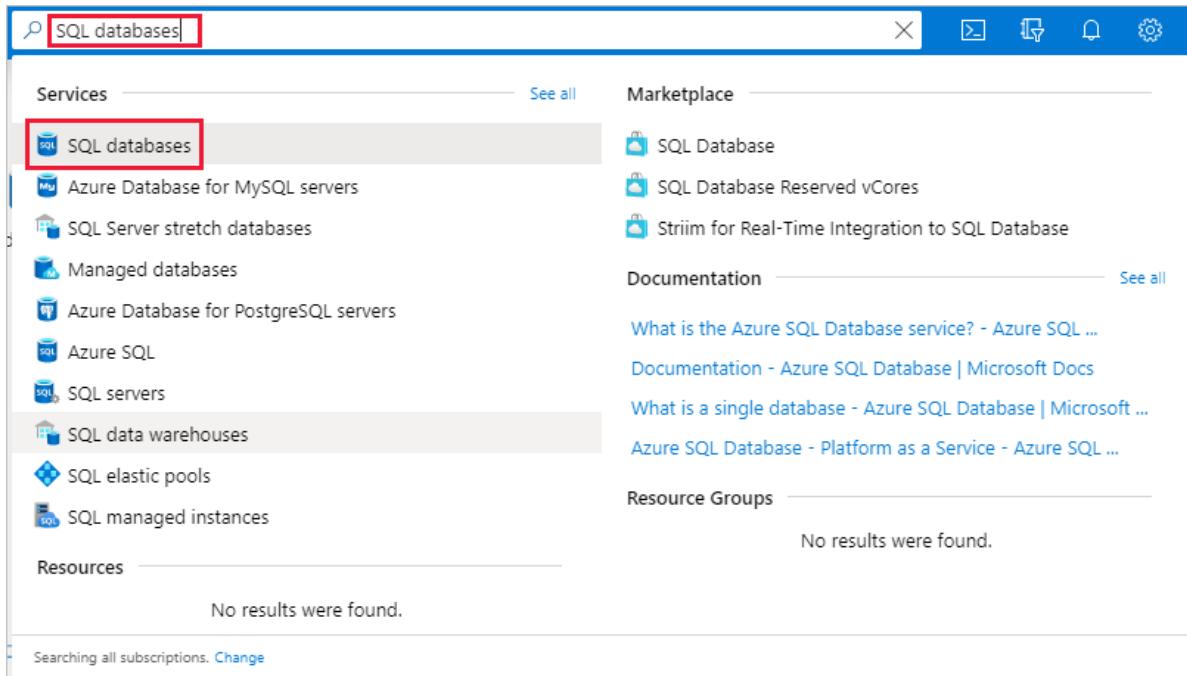
For PowerShell examples on how to configure SQL Data Sync, see [How to sync between Azure SQL databases](#) or [an Azure SQL Database and a SQL Server on-premises database](#)

## IMPORTANT

Azure SQL Data Sync does **not** support Azure SQL Database Managed Instance at this time.

## Create sync group

1. Go to the [Azure portal](#) to find your SQL database. Search for and select **SQL databases**.



The screenshot shows the Azure portal search interface. The search bar at the top contains the text "SQL databases". Below the search bar, there are two main sections: "Services" and "Marketplace". In the "Services" section, under the heading "See all", the "SQL databases" item is highlighted with a red box. Other items listed include "Azure Database for MySQL servers", "SQL Server stretch databases", "Managed databases", "Azure Database for PostgreSQL servers", "Azure SQL", "SQL servers", "SQL data warehouses", "SQL elastic pools", and "SQL managed instances". In the "Marketplace" section, there are links to "SQL Database", "SQL Database Reserved vCores", and "Striim for Real-Time Integration to SQL Database". Below the "Marketplace" section, there is a "Documentation" section with links to "What is the Azure SQL Database service?", "Documentation - Azure SQL Database | Microsoft Docs", "What is a single database - Azure SQL Database | Microsoft Docs", and "Azure SQL Database - Platform as a Service - Azure SQL Database | Microsoft Docs". At the bottom of the page, there is a "Resource Groups" section with the message "No results were found." and a note "Searching all subscriptions. Change".

2. Select the database you want to use as the hub database for Data Sync.

The screenshot shows the Microsoft Azure portal's 'SQL databases' blade. At the top, there's a search bar and a navigation bar with 'Microsoft Azure'. Below that is a table header with columns: Name, Status, Replication role, Server, Pricing tier, Location, and Subscription. There are five items listed:

Name	Status	Replication role	Server	Pricing tier	Location	Subscription
contoso-adventure	Online	None	contososql	General Purpose: Gen5, 2 v...	West US 2	Contoso Subscription
contoso-database	Online	None	contoso-sql-server	Basic	East US	Contoso Subscription
contoso-dest	Online	None	contososql	Standard S0: 10 DTU	West US 2	Contoso Subscription
contoso-new_server	Online	None	contososql	General Purpose: Gen5, 2 v...	West US 2	Contoso Subscription

#### NOTE

The hub database is a sync topology's central endpoint, in which a sync group has multiple database endpoints. All other member databases with endpoints in the sync group, sync with the hub database.

3. On the **SQL database** menu for the selected database, select **Sync to other databases**.

The screenshot shows the Azure portal interface for managing a SQL database. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Power Platform (Power BI (preview), PowerApps (preview), Flow (preview)), Settings (Configure, Geo-Replication, Connection strings), Sync to other databases (which is highlighted with a red box), Add Azure Search, Properties, Locks, and Export template.

The main content area is titled "Sync Group" and displays a message: "Name" and "You do not have any sync groups". Below this is the "Sync Agent" section, which also displays a "Name" field and the message "You do not have any agents".

- On the **Sync to other databases** page, select **New Sync Group**. The **New sync group** page opens with **Create sync group (step 1)** highlighted.

The screenshot shows the "Create Data Sync Group" dialog box. On the left, a navigation pane lists three steps: 1. Create sync group (Not Completed), 2. Add sync members (None Selected), and 3. Configure sync group (Not Complete). The main area is titled "Create Data Sync Group" and contains the following configuration fields:

- \* Sync Group Name:** A text input field containing a single character "I".
- Sync Metadata Database:** A radio button group where "New database" is selected (indicated by a blue circle).
- \* Create new database:** A link to "Configure database settings".
- \* Automatic Sync:** A switch button set to "On".
- \* Conflict Resolution:** A dropdown menu with an arrow pointing down.

A large red box highlights the "Sync Group Name" input field and the "Create new database" link. A smaller red box highlights the "OK" button at the bottom right of the dialog.

On the **Create Data Sync Group** page, change the following settings:

SETTING	DESCRIPTION
<b>Sync Group Name</b>	Enter a name for the new sync group. This name is distinct from the name of the database itself.
<b>Sync Metadata Database</b>	<p>Choose to create a database (recommended) or to use an existing database.</p> <p>If you choose <b>New database</b>, select <b>Create new database</b>. Then on the <b>SQL Database</b> page, name and configure the new database and select <b>OK</b>.</p> <p>If you choose <b>Use existing database</b>, select the database from the list.</p>
<b>Automatic Sync</b>	<p>Select <b>On</b> or <b>Off</b>.</p> <p>If you choose <b>On</b>, enter a number and select <b>Seconds</b>, <b>Minutes</b>, <b>Hours</b>, or <b>Days</b> in the <b>Sync Frequency</b> section.</p>
<b>Conflict Resolution</b>	<p>Select <b>Hub win</b> or <b>Member win</b>.</p> <p><b>Hub win</b> means when conflicts occur, data in the hub database overwrites conflicting data in the member database.</p> <p><b>Member win</b> means when conflicts occur, data in the member database overwrites conflicting data in the hub database.</p>

#### NOTE

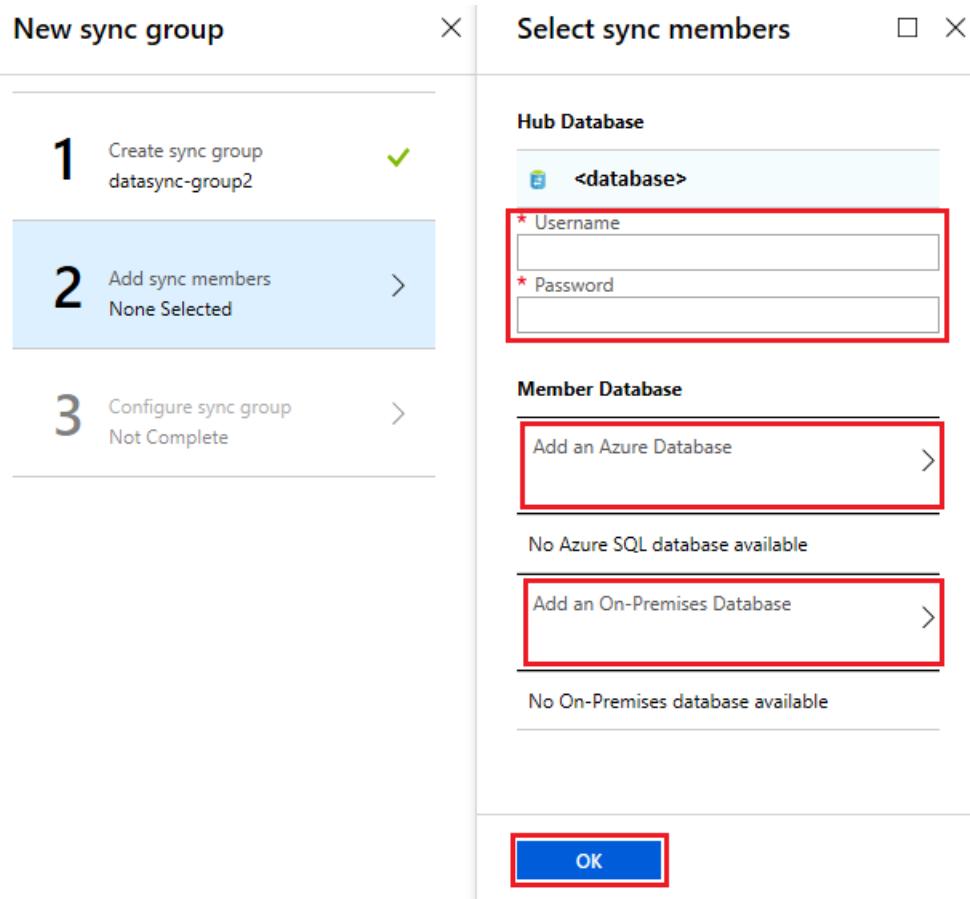
Microsoft recommends to create a new, empty database for use as the **Sync Metadata Database**. Data Sync creates tables in this database and runs a frequent workload. This database is shared as the **Sync Metadata Database** for all Sync Groups in a selected region and you can't change the database or its name without removing all Sync Groups and Sync Agents in the region.

Select **OK** and wait for the sync group to be created and deployed.

## Add sync members

After the new sync group is created and deployed, **Add sync members (step 2)** is highlighted on the **New sync group** page.

In the **Hub Database** section, enter existing credentials for the SQL Database server on which the hub database is located. Don't enter *new* credentials in this section.



#### To add an Azure SQL Database

In the **Member Database** section, optionally add an Azure SQL Database to the sync group by selecting **Add an Azure SQL Database**. The **Configure Azure SQL Database** page opens.

The 'Configure Azure Database' dialog box shows the following fields:

- \* Sync Member Name
- \* Subscription
- \* Azure SQL Server
- \* Azure SQL Database
- \* Sync Directions
- \* Username
- \* Password

The 'OK' button at the bottom is highlighted with a red box.

On the **Configure Azure SQL Database** page, change the following settings:

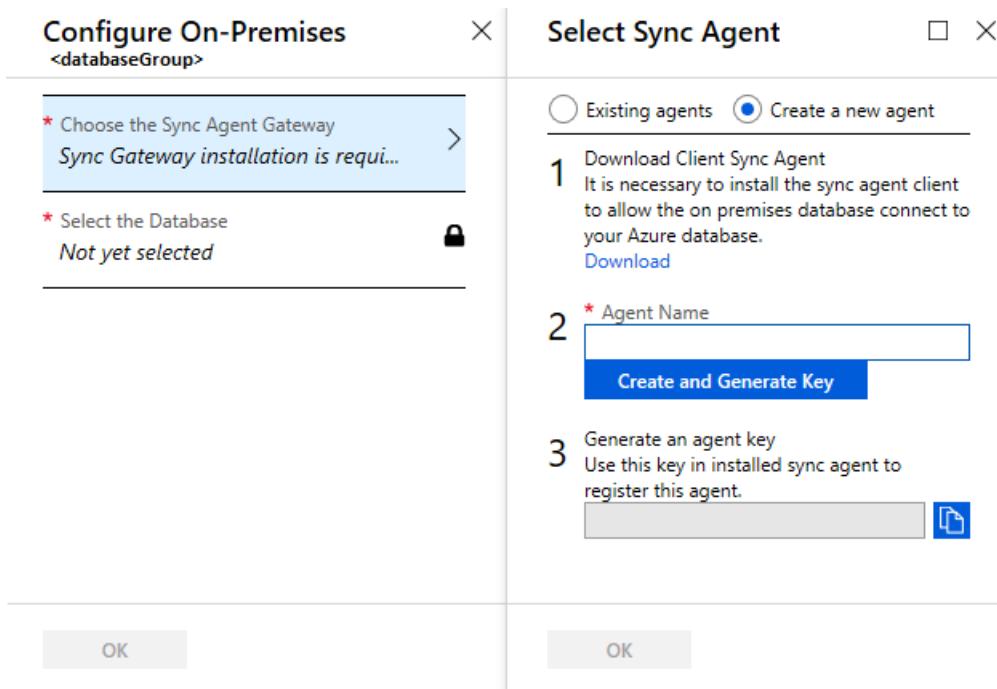
SETTING	DESCRIPTION
<b>Sync Member Name</b>	Provide a name for the new sync member. This name is distinct from the database name itself.
<b>Subscription</b>	Select the associated Azure subscription for billing purposes.
<b>Azure SQL Server</b>	Select the existing SQL Database server.
<b>Azure SQL Database</b>	Select the existing SQL database.
<b>Sync Directions</b>	Select <b>Bi-directional Sync</b> , <b>To the Hub</b> , or <b>From the Hub</b> .
<b>Username and Password</b>	Enter the existing credentials for the SQL Database server on which the member database is located. Don't enter <i>new</i> credentials in this section.

Select **OK** and wait for the new sync member to be created and deployed.

#### To add an on-premises SQL Server database

In the **Member Database** section, optionally add an on-premises SQL Server to the sync group by selecting **Add an On-Premises Database**. The **Configure On-Premises** page opens where you can do the following things:

1. Select **Choose the Sync Agent Gateway**. The **Select Sync Agent** page opens.



2. On the **Choose the Sync Agent** page, choose whether to use an existing agent or create an agent.

If you choose **Existing agents**, select the existing agent from the list.

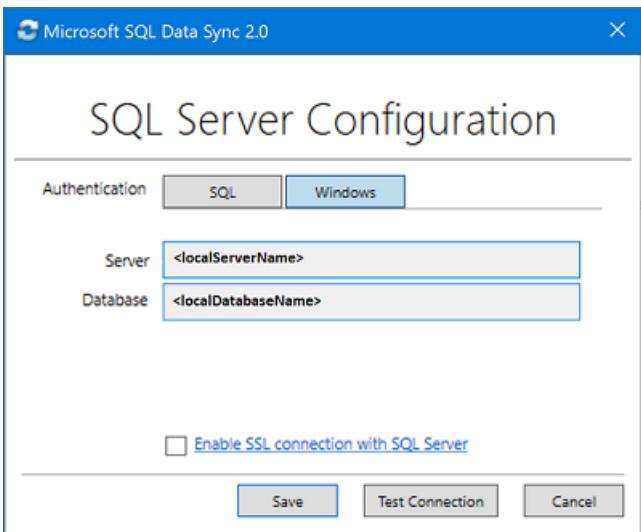
If you choose **Create a new agent**, do the following things:

- a. Download the data sync agent from the link provided and install it on the computer where the SQL Server is located. You can also download the agent directly from [SQL Azure Data Sync Agent](#).

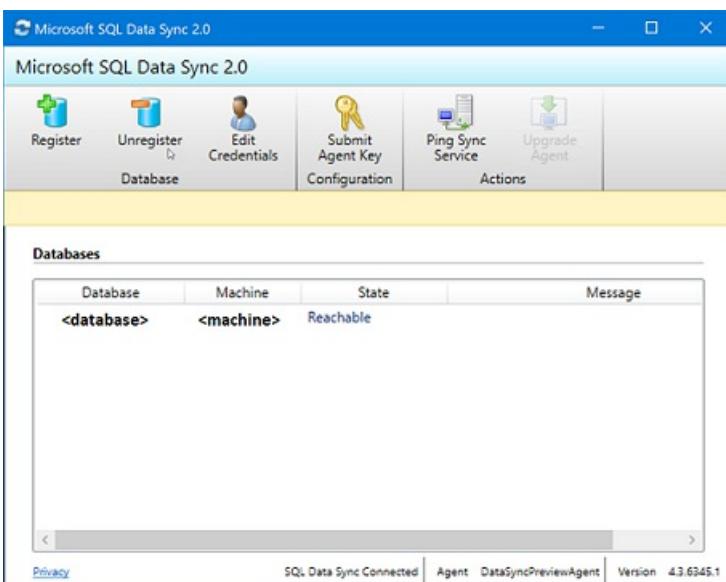
#### IMPORTANT

You have to open outbound TCP port 1433 in the firewall to let the client agent communicate with the server.

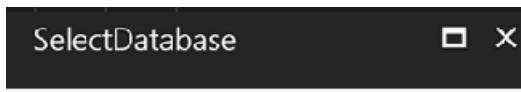
- b. Enter a name for the agent.
  - c. Select **Create and Generate Key** and copy the agent key to the clipboard.
  - d. Select **OK** to close the **Select Sync Agent** page.
3. On the SQL Server computer, locate and run the Client Sync Agent app.
- 
- a. In the sync agent app, select **Submit Agent Key**. The **Sync Metadata Database Configuration** dialog box opens.
- 
- c. Select **Register** to register a SQL Server database with the agent. The **SQL Server Configuration** dialog box opens.



- d. In the **SQL Server Configuration** dialog box, choose to connect using SQL Server authentication or Windows authentication. If you choose SQL Server authentication, enter the existing credentials. Provide the SQL Server name and the name of the database that you want to sync and select **Test connection** to test your settings. Then select **Save** and the registered database appears in the list.



- e. Close the Client Sync Agent app.
4. In the portal, on the **Configure On-Premises** page, select **Select the Database**.
5. On the **Select Database** page, in the **Sync Member Name** field, provide a name for the new sync member. This name is distinct from the name of the database itself. Select the database from the list. In the **Sync Directions** field, select **Bi-directional Sync**, **To the Hub**, or **From the Hub**.



6. Select **OK** to close the **Select Database** page. Then select **OK** to close the **Configure On-Premises** page and wait for the new sync member to be created and deployed. Finally, select **OK** to close the **Select sync members** page.

#### NOTE

To connect to SQL Data Sync and the local agent, add your user name to the role *DataSync\_Executor*. Data Sync creates this role on the SQL Server instance.

## Configure sync group

After the new sync group members are created and deployed, **Configure sync group (step 3)** is highlighted in the **New sync group** page.

The screenshot shows the 'New sync group' configuration page with three steps listed on the left:

- 1 Create sync group **datasync-group2** ✓
- 2 Add sync members 2 Selected ✓
- 3 Configure sync group Not Complete >

The 'Tables' tab is selected. On the right, there's a list of tables from the 'Hub Database':

NAME	COLUMNS	NAME	DATA TYPE	DESCRIPTIONS
dbo.BuildVersion	4	SalesLT.Product	int(4)	Primary Key
dbo.ErrorLog	9	ProductID	int(4)	
SalesLT.Address	9	Name	userdefineddatatype(50)	Unsupported
SalesLT.Customer	15	ProductNumber	nvarchar(25)	
SalesLT.CustomerAddr...	5	Color	nvarchar(15)	
<b>SalesLT.Product</b>	<b>17</b>	StandardCost	money(8)	
SalesLT.ProductCateg...	5	ListPrice	money(8)	
SalesLT.ProductDescri...	4	Size	nvarchar(5)	
SalesLT.ProductModel	5	Weight	decimal(5)	
SalesLT.ProductModel...	5	ProductCategoryID	int(4)	
SalesLT.SalesOrderDet...	9	ProductModelID	int(4)	
SalesLT.SalesOrderHe...	22	SellStartDate	datetime(8)	
		SellEndDate	datetime(8)	

A red box highlights the 'Save' button at the bottom of the table area.

1. On the **Tables** page, select a database from the list of sync group members and select **Refresh schema**.
2. From the list, select the tables you want to sync. By default, all columns are selected, so disable the checkbox for the columns you don't want to sync. Be sure to leave the primary key column selected.
3. Select **Save**.

4. By default, databases are not synced until scheduled or manually run. To run a manual sync, navigate to your SQL database in the Azure portal, select **Sync to other databases**, and select the sync group. The **Data Sync** page opens. Select **Sync**.

The screenshot shows the Azure portal's Data Sync interface. At the top, the navigation path is: Home > SQL databases > data-sync - Sync to other databases > <datasyncGroup>. Below the header, there are buttons for Sync (highlighted with a red box), Stop, Properties, Delete, Filter Logs, and Refresh Logs. The main area displays two sections: 'Databases' (containing 2 databases) and 'Tables' (containing 1 table). In the 'Logs' section, there is one entry:

TYPE	DATE/TIME	MEMBER DATABASE	DETAILS
Success	01/08/19, 04:42:29 PM	<database>	Schema information obtained successfully.

## FAQ

### How frequently can Data Sync synchronize my data?

The minimal duration between synchronizations is five minutes.

### Does SQL Data Sync fully create tables?

If sync schema tables are missing in the destination database, SQL Data Sync creates them with the columns you selected. However, this doesn't result in a full-fidelity schema for the following reasons:

- Only columns you select are created in the destination table. Columns not selected are ignored.
- Only selected column indexes are created in the destination table. For columns not selected, those indexes are ignored.
- Indexes on XML type columns aren't created.
- CHECK constraints aren't created.
- Triggers on the source tables aren't created.
- Views and stored procedures aren't created.

Because of these limitations, we recommend the following things:

- For production environments, create the full-fidelity schema yourself.
- When experimenting with the service, use the auto-provisioning feature.

### Why do I see tables I didn't create?

Data Sync creates additional tables in the database for change tracking. Don't delete these or Data Sync stops working.

### Is my data convergent after a sync?

Not necessarily. Take a sync group with a hub and three spokes (A, B, and C) where synchronizations are Hub to A, Hub to B, and Hub to C. If a change is made to database A *after* the Hub to A sync, that change isn't written to

database B or database C until the next sync task.

### How do I get schema changes into a sync group?

Make and propagate all schema changes manually.

1. Replicate the schema changes manually to the hub and to all sync members.
2. Update the sync schema.

For adding new tables and columns:

New tables and columns don't impact the current sync and Data Sync ignores them until they're added to the sync schema. When adding new database objects, follow the sequence:

1. Add new tables or columns to the hub and to all sync members.
2. Add new tables or columns to the sync schema.
3. Begin inserting values into the new tables and columns.

For changing the data type of a column:

When you change the data type of an existing column, Data Sync continues to work as long as the new values fit the original data type defined in the sync schema. For example, if you change the type in the source database from **int** to **bigint**, Data Sync continues to work until you insert a value too large for the **int** data type. To complete the change, replicate the schema change manually to the hub and to all sync members, then update the sync schema.

### How can I export and import a database with Data Sync?

After you export a database as a *.bacpac* file and import the file to create a database, do the following to use Data Sync in the new database:

1. Clean up the Data Sync objects and additional tables on the new database by using [this script](#). The script deletes all the required Data Sync objects from the database.
2. Recreate the sync group with the new database. If you no longer need the old sync group, delete it.

### Where can I find information on the client agent?

For frequently asked questions about the client agent, see [Agent FAQ](#).

## Next steps

Congratulations. You've created a sync group that includes both a SQL Database instance and a SQL Server database.

For more info about SQL Data Sync, see:

- [Data Sync Agent for Azure SQL Data Sync](#)
- [Best practices and How to troubleshoot issues with Azure SQL Data Sync](#)
- [Monitor SQL Data Sync with Azure Monitor logs](#)
- [Update the sync schema with Transact-SQL or PowerShell](#)

For more info about SQL Database, see:

- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Tutorial: Add an Azure SQL Database elastic pool to a failover group

12/30/2019 • 19 minutes to read • [Edit Online](#)

Configure a failover group for an Azure SQL Database elastic pool and test failover using the Azure portal. In this tutorial, you will learn how to:

- Create an Azure SQL Database single database.
- Add the single database into an elastic pool.
- Create a [failover group](#) for two elastic pools between two logical SQL servers.
- Test failover.

## Prerequisites

To complete this tutorial, make sure you have:

- An Azure subscription. [Create a free account](#) if you don't already have one.

## 1 - Create a single database

In this step, you will create an Azure SQL Database single database.

### IMPORTANT

Be sure to set up firewall rules to use the public IP address of the computer you're using to complete this article.

For information, see [Create a database-level firewall rule](#) or to determine the IP address used for the server-level firewall rule for your computer see [Create a server-level firewall](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Create your resource group and single database using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type *Azure SQL* in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select **+ Add** to open the **Select SQL deployment option** page. You can view additional information about the different databases by selecting **Show details** on the **Databases** tile.
3. Select **Create**:

The screenshot shows the 'Select SQL deployment option' page. It features three main sections: 'SQL databases' (best for modern cloud applications), 'SQL managed instances' (best for most migrations to the cloud), and 'SQL virtual machines' (best for migrations and applications requiring OS-level access). Each section includes a 'Create' button and a 'Show details' link.

4. On the **Basics** tab, in the **Project Details** section, type or select the following values:

- **Subscription:** Drop down and select the correct subscription, if it doesn't appear.
- **Resource group:** Select **Create new**, type `myResourceGroup`, and select **OK**.

The screenshot shows the 'Create SQL Database' Basics tab. It includes a 'Project details' section where users can select a subscription and a resource group. The 'Subscription' dropdown is highlighted with a purple border, and the 'Resource group' dropdown also has a purple border, indicating they are the focus of the step. A note at the top says: 'Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize.' A 'Learn more' link is also present.

5. In the **Database Details** section, type or select the following values:

- **Database name:** Enter `mySampleDatabase`.
- **Server:** Select **Create new**, enter the following values and then select **Select**.
  - **Server name:** Type `mysqlserver`; along with some numbers for uniqueness.
  - **Server admin login:** Type `azureuser`.
  - **Password:** Type a complex password that meets password requirements.
  - **Location:** Choose a location from the drop-down, such as `West US`.

New server

\* Server name  
mysqlserver .database.windows.net

\* Server admin login  
azureuser

\* Password  
.....

\* Confirm password  
.....

\* Location  
West US

Allow Azure services to access server

#### IMPORTANT

Remember to record the server admin login and password so you can log in to the server and databases for this and other quickstarts. If you forget your login or password, you can get the login name or reset the password on the **SQL server** page. To open the **SQL server** page, select the server name on the database **Overview** page after database creation.

- **Want to use SQL elastic pool:** Select the **No** option.
- **Compute + storage:** Select **Configure database**.

DATABASE DETAILS

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

\* Database name  
mySampleDatabase

\* Server [\(new\) mysqlserver \( West US \)](#)  
[Create new](#)

\* Want to use SQL elastic pool? [?](#)  Yes  No

\* Compute + storage [?](#)

<b>Standard S0</b>
10 DTUs, 1 GB storage
<a href="#">Configure database</a>

\* Resource group [?](#)  
(New) myResourceGroup

- Select **Provisioned**.

- Review the settings for **vCores**, and **Data max size**. Change these as desired.

- Optionally, you can also select **Change configuration** to change the hardware generation.

- Select **Apply**.

6. Select the **Networking** tab and decide if you want to [Allow Azure services and resources to access this server](#), or add a [private endpoint](#).

Name	Subscription
Click on add to create private endpoint	

7. Select the **Additional settings** tab.
8. In the **Data source** section, under **Use existing data**, select **Sample**.

Home > Azure SQL > Select SQL deployment option > Create SQL Database

## Create SQL Database

Microsoft

Basics Networking Additional settings Tags Review + create

Customize additional configuration parameters including collation & sample data.

**Data source**

Start with a blank database, restore from a backup or select sample data to populate your new database.

Use existing data \*

None Backup Sample

AdventureWorksLT will be created as the sample database.

**Database collation**

Database collation defines the rules that sort and compare data, and cannot be changed after database creation. The default database collation is SQL\_Latin1\_General\_CI\_AS. [Learn more](#)

Collation ⓘ SQL\_Latin1\_General\_CI\_AS

**Advanced data security**

Protect your data using advanced data security, a unified security package including data classification, vulnerability assessment and advanced threat protection for your server. [Learn more](#)

**Advanced data security has already been enabled on the selected server.**

### IMPORTANT

Make sure to select the **Sample (AdventureWorksLT)** data so you can follow easily this and other Azure SQL Database quickstarts that use this data.

9. Leave the rest of the values as default and select **Review + Create** at the bottom of the form.
10. Review the final settings and select **Create**.
11. On the **SQL Database** form, select **Create** to deploy and provision the resource group, server, and database.

## 2 - Add single database to elastic pool

In this step, you will create an elastic pool, and add your single database to it.

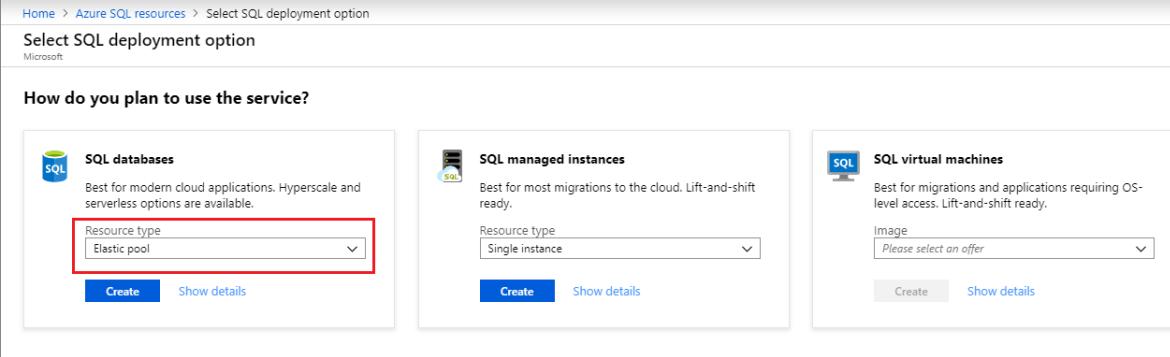
- [Portal](#)
- [PowerShell](#)

Create your elastic pool using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the Azure portal. If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it

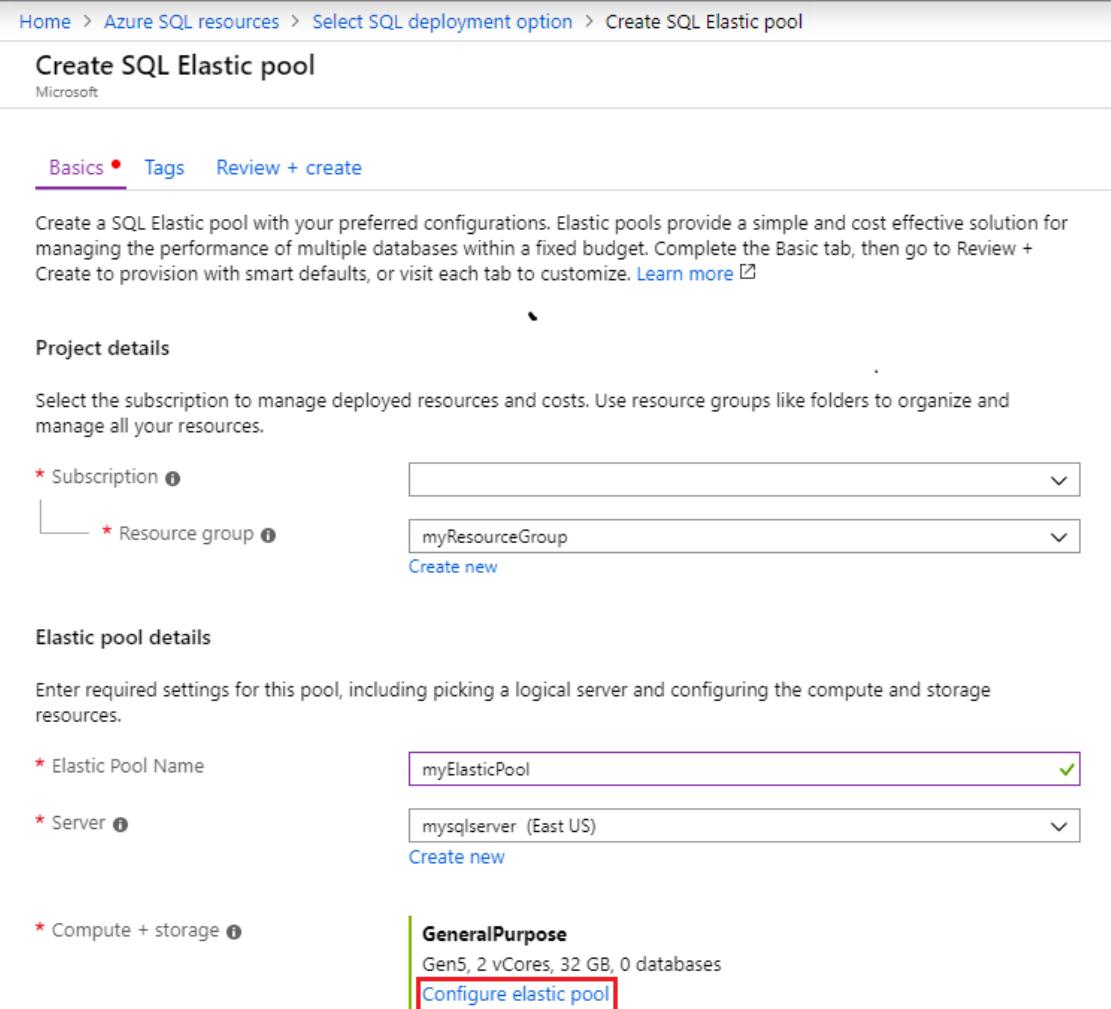
and add it as an item in the left-hand navigation.

2. Select **+ Add** to open the **Select SQL deployment option** page. You can view additional information about the different databases by selecting Show details on the Databases tile.
3. Select **Elastic pool** from the **Resource type** drop-down in the **SQL Databases** tile. Select **Create** to create your elastic pool.



4. Configure your elastic pool with the following values:

- **Name:** Provide a unique name for your elastic pool, such as `myElasticPool`.
- **Subscription:** Select your subscription from the drop-down.
- **ResourceGroup:** Select `myResourceGroup` from the drop-down, the resource group you created in section 1.
- **Server:** Select the server you created in section 1 from the drop-down.



- **Compute + storage:** Select **Configure elastic pool** to configure your compute, storage, and add your single database to your elastic pool. On the **Pool Settings** tab, leave the default of Gen5, with 2 vCores and 32gb.
5. On the **Configure** page, select the **Databases** tab, and then choose to **Add database**. Choose the database you created in section 1 and then select **Apply** to add it to your elastic pool. Select **Apply** again to apply your elastic pool settings and close the **Configure** page.

The screenshot shows the 'Configure' page for creating an elastic pool. The 'Databases' tab is active. A red box highlights the '+ Add databases' button. Another red box highlights the 'mySampleDatabase' entry in the list of databases.

6. Select **Review + create** to review your elastic pool settings and then select **Create** to create your elastic pool.

### 3 - Create the failover group

In this step, you will create a **failover group** between an existing Azure SQL server and a new Azure SQL server in another region. Then add the elastic pool to the failover group.

- [Portal](#)
- [PowerShell](#)

Create your failover group using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select the elastic pool created in the previous section, such as `myElasticPool`.
3. On the **Overview** pane, select the name of the server under **Server name** to open the settings for the server.

The screenshot shows the 'Overview' page for the 'myElasticPool' elastic pool. The 'Server name' field is highlighted with a red box.

4. Select **Failover groups** under the **Settings** pane, and then select **Add group** to create a new failover group.

mysqlserver - Failover groups

**Add group**

Failover group are a SQL server feature designed to automatically

NAME	PRIMARY SERVER
You have no group created	

5. On the **Failover Group** page, enter or select the following values, and then select **Create**:

- **Failover group name:** Type in a unique failover group name, such as `failovergrouptutorial`.
- **Secondary server:** Select the option to *configure required settings* and then choose to **Create a new server**. Alternatively, you can choose an already-existing server as the secondary server. After entering the following values for your new secondary server, select **Select**.
  - **Server name:** Type in a unique name for the secondary server, such as `mysqlsecondary`.
  - **Server admin login:** Type `azureuser`
  - **Password:** Type a complex password that meets password requirements.
  - **Location:** Choose a location from the drop-down, such as `East US`. This location cannot be the same location as your primary server.

#### NOTE

The server login and firewall settings must match that of your primary server.

Failover group

Server

New server

Create a failover group to automatically failover databases in it.

\* Failover group name: failovergrouptutorial .database.windows.net

\* Secondary server: Configure required settings >

Read/Write failover policy: Automatic

Read/Write grace period (hours): 1 hours

Database within the group: Select databases to add

Create a new server

\* Server name: mysqlsecondary .database.windows.net

\* Server admin login: azureuser

\* Password: [REDACTED]

\* Confirm password: [REDACTED]

\* Location: East US

Allow Azure services to access server

6. Select **Databases within the group** then select the elastic pool you created in section 2. A warning should appear, prompting you to create an elastic pool on the secondary server. Select the warning, and then select **OK** to create the elastic pool on the secondary server.

The screenshot shows the Azure portal interface for creating a failover group. On the left, the 'Failover group' pane shows fields for 'Failover group name' (failovergroupTutorial), 'Secondary server' (mysqlsecondary (East US)), 'Read/Write failover policy' (Automatic), and 'Read/Write grace period (hours)' (1 hours). A red box highlights the 'Select databases to add' button (labeled 1).

In the center, the 'Databases' pane lists 'myElasticPool' (2 databases) and 'mySampleDatabase' (Primary, mysqlsecondary, Online). A red box highlights the 'Selected/Eligible databases' count (0/1).

On the right, the 'Elastic pool' pane shows 'myElasticPool' settings: Name (myElasticPool), Subscription (myResourceGroup), Server (mysqlsecondary (East US)), Pricing tier (GeneralPurpose Pool), and Configured (1 eDTU, 0 databases). A red box highlights the 'OK' button.

- Select **Select** to apply your elastic pool settings to the failover group, and then select **Create** to create your failover group. Adding the elastic pool to the failover group will automatically start the geo-replication process.

## 4 - Test failover

In this step, you will fail your failover group over to the secondary server, and then fail back using the Azure portal.

- [Portal](#)
- [PowerShell](#)

Test failover of your failover group using the Azure portal.

- Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
- Select the elastic pool created in the previous section, such as `myElasticPool`.
- Select the name of the server under **Server name** to open the settings for the server.

The screenshot shows the 'myElasticPool (mysqlserver/myElasticPool)' settings page. The 'Overview' tab is selected. The 'Server name' field is highlighted with a red box and contains the value 'mysqlserver.database.windows.net'. Other visible information includes 'Resource group (change) : myResourceGroup', 'Status : Ready', 'Location : East US', 'Resource configuration ... : GeneralPurpose: Gen5, 2 vCores', and 'Elastic databases : 0 databases'.

- Select **Failover groups** under the **Settings** pane and then choose the failover group you created in section 2.

Home > Azure SQL resources > myElasticPool (mysqlserver/myElasticPool) > mysqlserver - Failover groups

## mysqlserver - Failover groups

**Add group** **Refresh**

Failover group are a SQL server feature designed to automatically manage replication, connection, and failover between primary and secondary servers.

NAME	PRIMARY SERVER	SECONDARY SERVER
failovergrouptutorial	mysqlserver	mysqlsecondary

Search (Ctrl+ /)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Settings
- Quick start
- Failover groups**

- Review which server is primary, and which server is secondary.
- Select **Failover** from the task pane to fail over your failover group containing your elastic pool.
- Select **Yes** on the warning that notifies you that TDS sessions will be disconnected.

Home > Azure SQL resources > myElasticPool (mysqlserver/myElasticPool) > mysqlserver - Failover groups > failovergrouptutorial

### failovergrouptutorial

failover-sqlserver

**Save** **Discard** **Add databases** **Edit configuration** **Remove databases** **Failover** **Forced Failover** **Delete**

Configuration details Databases within group Databases selected to be added (0) Databases selected for removal (0)

SERVER	ROLE	READ/WRITE FAILOVER POLICY
mysqlserver (West US)	Primary	Automatic
mysqlsecondary (East US)	Secondary	

- Review which server is primary, which server is secondary. If failover succeeded, the two servers should have swapped roles.
- Select **Failover** again to fail the failover group back to the original settings.

## Clean up resources

Clean up resources by deleting the resource group.

- Portal
- PowerShell

1. Navigate to your resource group in the [Azure portal](#).
2. Select **Delete resource group** to delete all the resources in the group, as well as the resource group itself.
3. Type the name of the resource group, `myResourceGroup`, in the textbox, and then select **Delete** to delete the resource group.

**IMPORTANT**

If you want to keep the resource group but delete the secondary database, remove it from the failover group before deleting it. Deleting a secondary database before it is removed from the failover group can cause unpredictable behavior.

## Full script

- [PowerShell](#)
- [Portal](#)

There are no scripts available for the Azure portal.

## Next steps

In this tutorial, you added an Azure SQL Database elastic pool to a failover group, and tested failover. You learned how to:

- Create an Azure SQL Database single database.
- Add the single database into an elastic pool.
- Create a [failover group](#) for two elastic pools between two logical SQL servers.
- Test failover.

Advance to the next tutorial on how to migrate using DMS.

[Tutorial: Migrate SQL Server to a pooled database using DMS](#)

# Tutorial: Secure a single or pooled database

11/7/2019 • 11 minutes to read • [Edit Online](#)

In this tutorial you learn how to:

- Create server-level and database-level firewall rules
- Configure an Azure Active Directory (AD) administrator
- Manage user access with SQL authentication, Azure AD authentication, and secure connection strings
- Enable security features, such as advanced data security, auditing, data masking, and encryption

Azure SQL Database secures data in a single or pooled database by allowing you to:

- Limit access using firewall rules
- Use authentication mechanisms that require identity
- Use authorization with role-based memberships and permissions
- Enable security features

## NOTE

An Azure SQL database on a managed instance is secured using network security rules and private endpoints as described in [Azure SQL database managed instance](#) and [connectivity architecture](#).

To learn more, see the [Azure SQL Database security overview](#) and [capabilities](#) articles.

## TIP

The following Microsoft Learn module helps you learn for free about how to [Secure your Azure SQL Database](#).

## Prerequisites

To complete the tutorial, make sure you have the following prerequisites:

- [SQL Server Management Studio](#)
- An Azure SQL server and database
  - Create them with [Azure portal](#), [CLI](#), or [PowerShell](#)

If you don't have an Azure subscription, [create a free account](#) before you begin.

## Sign in to the Azure portal

For all steps in the tutorial, sign in to [Azure portal](#)

## Create firewall rules

SQL databases are protected by firewalls in Azure. By default, all connections to the server and database are rejected. To learn more, see [Azure SQL Database server-level and database-level firewall rules](#).

Set **Allow access to Azure services** to **OFF** for the most secure configuration. Then, create a [reserved IP \(classic deployment\)](#) for the resource that needs to connect, such as an Azure VM or cloud service, and only allow that IP address access through the firewall. If you're using the [resource manager](#) deployment model, a dedicated public IP

address is required for each resource.

#### NOTE

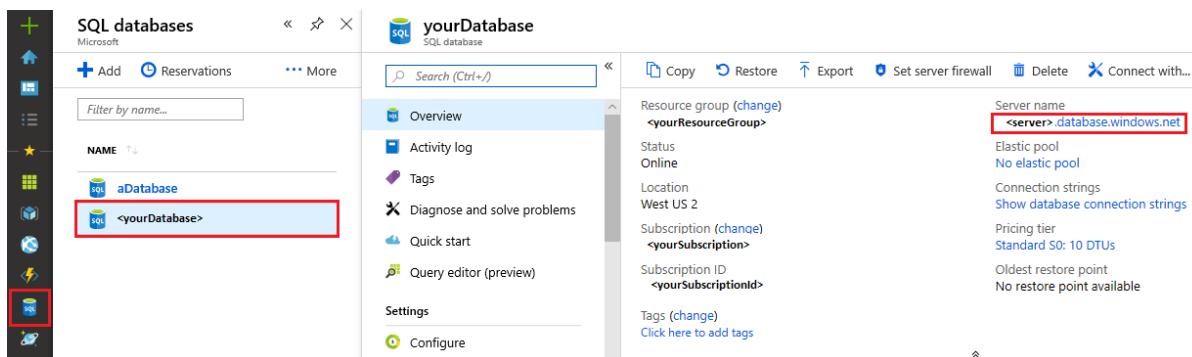
SQL Database communicates over port 1433. If you're trying to connect from within a corporate network, outbound traffic over port 1433 may not be allowed by your network's firewall. If so, you can't connect to the Azure SQL Database server unless your administrator opens port 1433.

## Set up SQL Database server firewall rules

Server-level IP firewall rules apply to all databases within the same SQL Database server.

To set up a server-level firewall rule:

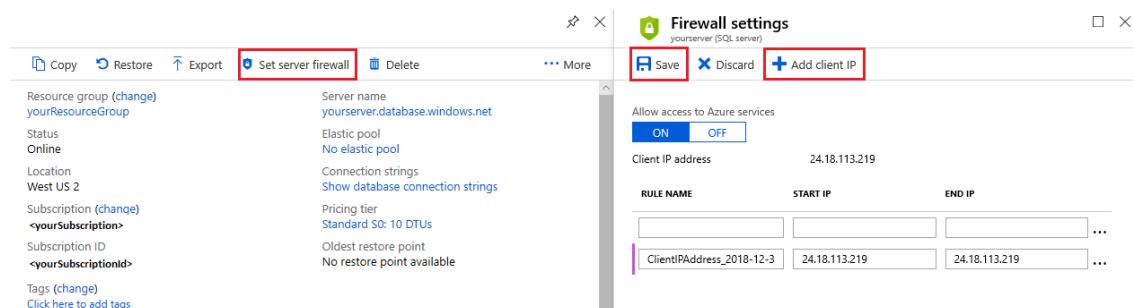
1. In Azure portal, select **SQL databases** from the left-hand menu, and select your database on the **SQL databases** page.



#### NOTE

Be sure to copy your fully qualified server name (such as *yourserver.database.windows.net*) for use later in the tutorial.

2. On the **Overview** page, select **Set server firewall**. The **Firewall settings** page for the database server opens.
  - a. Select **Add client IP** on the toolbar to add your current IP address to a new firewall rule. The rule can open port 1433 for a single IP address or a range of IP addresses. Select **Save**.



- b. Select **OK** and close the **Firewall settings** page.

You can now connect to any database in the server with the specified IP address or IP address range.

## Setup database firewall rules

Database-level firewall rules only apply to individual databases. The database will retain these rules during a server failover. Database-level firewall rules can only be configured using Transact-SQL (T-SQL) statements, and only after you've configured a server-level firewall rule.

To setup a database-level firewall rule:

1. Connect to the database, for example using [SQL Server Management Studio](#).
2. In **Object Explorer**, right-click the database and select **New Query**.
3. In the query window, add this statement and modify the IP address to your public IP address:

```
EXECUTE sp_set_database_firewall_rule N'Example DB Rule','0.0.0.4','0.0.0.4';
```

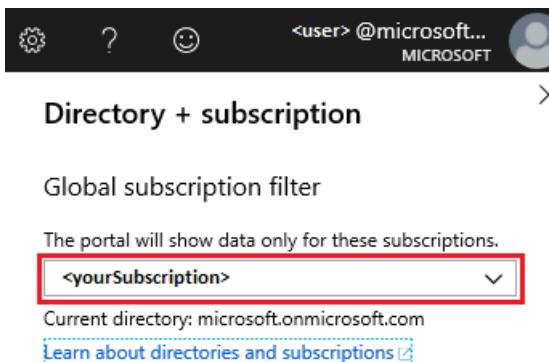
4. On the toolbar, select **Execute** to create the firewall rule.

**NOTE**

You can also create a server-level firewall rule in SSMS by using the [sp\\_set\\_firewall\\_rule](#) command, though you must be connected to the *master* database.

## Create an Azure AD admin

Make sure you're using the appropriate Azure Active Directory (AD) managed domain. To select the AD domain, use the upper-right corner of the Azure portal. This process confirms the same subscription is used for both Azure AD and the SQL Server hosting your Azure SQL database or data warehouse.



The portal will show data only for these subscriptions.  
<yourSubscription>  
Current directory: microsoft.onmicrosoft.com  
[Learn about directories and subscriptions](#)

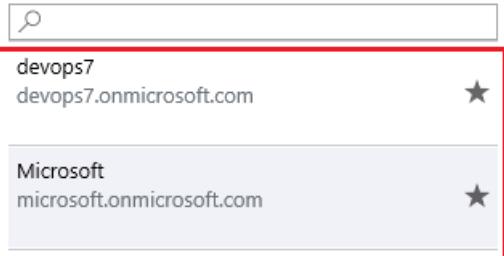
### Switch directory

Set your default directory

Sign in to your last visited directory

Favorites All Directories

A to Z ↑



devops7	★
devops7.onmicrosoft.com	
Microsoft	★
microsoft.onmicrosoft.com	

To set the Azure AD administrator:

1. In Azure portal, on the **SQL server** page, select **Active Directory admin**. Next select **Set admin**.

yourserver - Active Directory admin

SQL server

Search (Ctrl+ /) Set admin Remove admin Save

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Settings Quick start Failover groups Manage Backups Active Directory admin SQL databases

Azure Active Directory authentication allows you to centrally manage identity and access to your Azure SQL Database V12. [Learn more](#)

Active Directory admin ⓘ No Active Directory admin

**IMPORTANT**

You need to be either a "Company Administrator" or "Global Administrator" to perform this task.

- On the **Add admin** page, search and select the AD user or group and choose **Select**. All members and groups of your Active Directory are listed, and entries grayed out are not supported as Azure AD administrators. See [Azure AD features and limitations](#).

Add admin

Active Directory admin

+ Invite

Select ⓘ

<user>

MS <user> @microsoft.com

Selected

<user>

Select

**IMPORTANT**

Role-based access control (RBAC) only applies to the portal and isn't propagated to SQL Server.

- At the top of the **Active Directory admin** page, select **Save**.

The process of changing an administrator may take several minutes. The new administrator will appear in the **Active Directory admin** box.

#### **NOTE**

When setting an Azure AD admin, the new admin name (user or group) cannot exist as a SQL Server authentication user in the *master* database. If present, the setup will fail and roll back changes, indicating that such an admin name already exists. Since the SQL Server authentication user is not part of Azure AD, any effort to connect the user using Azure AD authentication fails.

For information about configuring Azure AD, see:

- [Integrate your on-premises identities with Azure AD](#)
- [Add your own domain name to Azure AD](#)
- [Microsoft Azure now supports federation with Windows Server AD](#)
- [Administer your Azure AD directory](#)
- [Manage Azure AD using PowerShell](#)
- [Hybrid identity required ports and protocols](#)

## Manage database access

Manage database access by adding users to the database, or allowing user access with secure connection strings. Connection strings are useful for external applications. To learn more, see [Azure SQL access control](#) and [AD authentication](#).

To add users, choose the database authentication type:

- **SQL authentication**, use a username and password for logins and are only valid in the context of a specific database within the server
- **Azure AD authentication**, use identities managed by Azure AD

#### **SQL authentication**

To add a user with SQL authentication:

1. Connect to the database, for example using [SQL Server Management Studio](#).
2. In **Object Explorer**, right-click the database and choose **New Query**.
3. In the query window, enter the following command:

```
CREATE USER ApplicationUser WITH PASSWORD = 'YourStrongPassword1';
```

4. On the toolbar, select **Execute** to create the user.
5. By default, the user can connect to the database, but has no permissions to read or write data. To grant these permissions, execute the following commands in a new query window:

```
ALTER ROLE db_datareader ADD MEMBER ApplicationUser;
ALTER ROLE db_datawriter ADD MEMBER ApplicationUser;
```

#### **NOTE**

Create non-administrator accounts at the database level, unless they need to execute administrator tasks like creating new users.

## Azure AD authentication

Azure Active Directory authentication requires that database users are created as contained. A contained database user maps to an identity in the Azure AD directory associated with the database and has no login in the *master* database. The Azure AD identity can either be for an individual user or a group. For more information, see [Contained database users, make your database portable](#) and review the [Azure AD tutorial](#) on how to authenticate using Azure AD.

#### NOTE

Database users (excluding administrators) cannot be created using the Azure portal. Azure RBAC roles do not propagate to SQL servers, databases, or data warehouses. They are only used to manage Azure resources and do not apply to database permissions.

For example, the *SQL Server Contributor* role does not grant access to connect to a database or data warehouse. This permission must be granted within the database using T-SQL statements.

#### IMPORTANT

Special characters like colon `:` or ampersand `&` are not supported in user names in the T-SQL `CREATE LOGIN` and `CREATE USER` statements.

To add a user with Azure AD authentication:

1. Connect to your Azure SQL server using an Azure AD account with at least the *ALTER ANY USER* permission.
2. In **Object Explorer**, right-click the database and select **New Query**.
3. In the query window, enter the following command and modify `<Azure_AD_principal_name>` to the principal name of the Azure AD user or the display name of the Azure AD group:

```
CREATE USER <Azure_AD_principal_name> FROM EXTERNAL PROVIDER;
```

#### NOTE

Azure AD users are marked in the database metadata with type `E (EXTERNAL_USER)` and type `X (EXTERNAL_GROUPS)` for groups. For more information, see [sys.database\\_principals](#).

## Secure connection strings

To ensure a secure, encrypted connection between the client application and SQL database, a connection string must be configured to:

- Request an encrypted connection
- Not trust the server certificate

The connection is established using Transport Layer Security (TLS) and reduces the risk of a man-in-the-middle attack. Connection strings are available per database and are pre-configured to support client drivers such as ADO.NET, JDBC, ODBC, and PHP. For information about TLS and connectivity, see [TLS considerations](#).

To copy a secure connection string:

1. In Azure portal, select **SQL databases** from the left-hand menu, and select your database on the **SQL databases** page.
2. On the **Overview** page, select **Show database connection strings**.

3. Select a driver tab and copy the complete connection string.

ADO.NET    JDBC    ODBC    PHP

ADO.NET (SQL authentication)

```
Server=tcp:yourserver.database.windows.net,1433;Initial Catalog=yourDatabase;Persist Security Info=False;User ID={your_username};Password={your_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;
```



[Download ADO.NET driver for SQL server](#)

## Enable security features

Azure SQL Database provides security features that are accessed using the Azure portal. These features are available for both the database and server, except for data masking, which is only available on the database. To learn more, see [Advanced data security](#), [Auditing](#), [Dynamic data masking](#), and [Transparent data encryption](#).

### Advanced data security

The advanced data security feature detects potential threats as they occur and provides security alerts on anomalous activities. Users can explore these suspicious events using the auditing feature, and determine if the event was to access, breach, or exploit data in the database. Users are also provided a security overview that includes a vulnerability assessment and the data discovery and classification tool.

#### NOTE

An example threat is SQL injection, a process where attackers inject malicious SQL into application inputs. An application can then unknowingly execute the malicious SQL and allow attackers access to breach or modify data in the database.

To enable advanced data security:

1. In Azure portal, select **SQL databases** from the left-hand menu, and select your database on the **SQL databases** page.
2. On the **Overview** page, select the **Server name** link. The database server page will open.
3. On the **SQL server** page, find the **Security** section and select **Advanced Data Security**.
  - a. Select **ON** under **Advanced Data Security** to enable the feature. Choose a storage account for saving vulnerability assessment results. Then select **Save**.

Save
 Discard
 Feedback

Advanced Threat Protection

**ON** **OFF**

### THREAT DETECTION SETTINGS

Send alerts to ?

Email addresses

Email service and co-administrators

---

Storage details >

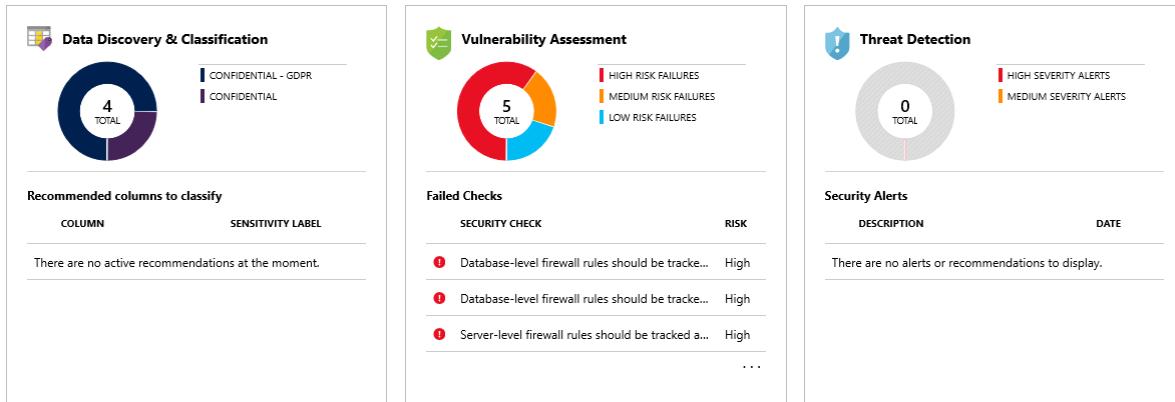
---

Threat Detection types >

All

You can also configure emails to receive security alerts, storage details, and threat detection types.

4. Return to the **SQL databases** page of your database and select **Advanced Data Security** under the **Security** section. Here you'll find various security indicators available for the database.



If anomalous activities are detected, you receive an email with information on the event. This includes the nature of the activity, database, server, event time, possible causes, and recommended actions to investigate and mitigate the potential threat. If such an email is received, select the **Azure SQL Auditing Log** link to launch the Azure portal and show relevant auditing records for the time of the event.

⚠ Potential Sql Injection was detected on your database demo-db on server demo-server.sqltest-eg1.mscds.com starting at 2015-10-29 08:46:58.

A database principal from IP 'xx.xxx.xxx.xxx' has executed a number of queries that match a SQL injection pattern including a query.  
Potential causes: SQL Injection, Penetration testing.  
It is recommended to review any code that constructs SQL statements for injection vulnerabilities.

You can inspect the [Azure SQL DB Audit Logs](#) around the time of the event.  
The first associated audit record has eventID '48ce7f75-3f02-49a0-8b11-561a176e6ee4'.

Your [feedback](#) is highly appreciated and will help us to improve our detection capabilities. If you would like to provide feedback, please reply to this email.

## Auditing

The auditing feature tracks database events and writes events to an audit log in either Azure storage, Azure Monitor logs, or to an event hub. Auditing helps maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate potential security violations.

To enable auditing:

1. In Azure portal, select **SQL databases** from the left-hand menu, and select your database on the **SQL databases** page.
2. In the **Security** section, select **Auditing**.
3. Under **Auditing** settings, set the following values:
  - a. Set **Auditing** to **ON**.
  - b. Select **Audit log destination** as any of the following:
    - **Storage**, an Azure storage account where event logs are saved and can be downloaded as .xel files

### TIP

Use the same storage account for all audited databases to get the most from auditing report templates.

- **Log Analytics**, which automatically stores events for query or further analysis

### NOTE

A **Log Analytics workspace** is required to support advanced features such as analytics, custom alert rules, and Excel or Power BI exports. Without a workspace, only the query editor is available.

- **Event Hub**, which allows events to be routed for use in other applications

c. Select **Save**.

Auditing

**ON** OFF

Audit log destination (choose at least one):

- Storage
- Log Analytics
- Event Hub

4. Now you can select **View audit logs** to view database events data.

EVENT TIME (UTC)	PRINCIPAL NAME	EVENT TYPE	ACTION STATUS
12/20/2018 12:12:28 AM	<user>@microsoft.com	BATCH COMPLETED	Succeeded
12/20/2018 12:12:28 AM	<user>@microsoft.com	DATABASE AUTHENTICATION SUCCEEDED	Succeeded
12/20/2018 12:11:12 AM	ApplicationUser	DATABASE AUTHENTICATION FAILED	Failed

[Load more](#)

#### IMPORTANT

See [SQL database auditing](#) on how to further customize audit events using PowerShell or REST API.

#### Dynamic data masking

The data masking feature will automatically hide sensitive data in your database.

To enable data masking:

1. In Azure portal, select **SQL databases** from the left-hand menu, and select your database on the **SQL databases** page.

2. In the **Security** section, select **Dynamic Data Masking**.
3. Under **Dynamic data masking** settings, select **Add mask** to add a masking rule. Azure will automatically populate available database schemas, tables, and columns to choose from.

MASK NAME	MASK FUNCTION
dbo_Person_LastName	Default value (0, xxxx, 01-01-1900)

SQL users excluded from masking (administrators are always excluded)

SCHEMA	TABLE	COLUMN	
dbo	Person	FirstName	<b>Add mask</b>
dbo	Student	Email	<b>Add mask</b>

4. Select **Save**. The selected information is now masked for privacy.

PersonId	FirstName	MiddleInitial	LastName	DateOfBirth
1	customerFirst	M	xxxx	2000-01-01

## Transparent data encryption

The encryption feature automatically encrypts your data at rest, and requires no changes to applications accessing the encrypted database. For new databases, encryption is on by default. You can also encrypt data using SSMS and the [Always encrypted](#) feature.

To enable or verify encryption:

1. In Azure portal, select **SQL databases** from the left-hand menu, and select your database on the **SQL databases** page.
2. In the **Security** section, select **Transparent data encryption**.
3. If necessary, set **Data encryption** to **ON**. Select **Save**.



Encrypts your databases, backups, and logs at rest without any changes to your application. To enable TDE, go to each database.

[Learn more](#)

#### Data encryption

#### Encryption status

 Encrypted

#### NOTE

To view encryption status, connect to the database using [SSMS](#) and query the `encryption_state` column of the [sys.dm\\_database\\_encryption\\_keys](#) view. A state of `3` indicates the database is encrypted.

## Next steps

In this tutorial, you've learned to improve the security of your database with just a few simple steps. You learned how to:

- Create server-level and database-level firewall rules
- Configure an Azure Active Directory (AD) administrator
- Manage user access with SQL authentication, Azure AD authentication, and secure connection strings
- Enable security features, such as advanced data security, auditing, data masking, and encryption

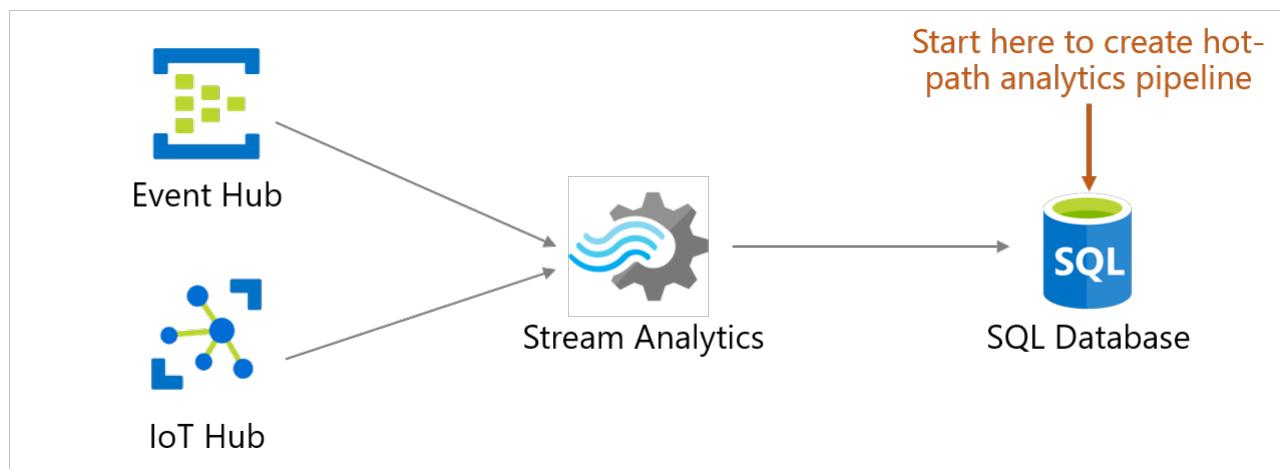
Advance to the next tutorial to learn how to implement geo-distribution.

[Implement a geo-distributed database](#)

# Stream data by using Azure SQL Database Stream Analytics integration (preview)

11/7/2019 • 5 minutes to read • [Edit Online](#)

Users can now ingest, process, view, and analyze real-time streaming data into a table directly from a SQL database in the Azure portal using [Azure Stream Analytics](#). This experience enables a wide variety of scenarios such as connected car, remote monitoring, fraud detection, and many more. In the Azure portal, you can select an events source (Event Hub/IoT Hub), view incoming real-time events, and select a table to store events. You can also write Stream Analytics Query Language queries in the portal to transform incoming events and store them in the selected table. This new entry point is in addition to the creation and configuration experiences that already exist in Stream Analytics. This experience starts from the context of your database, enabling you to quickly set up a Stream Analytics job and navigate seamlessly between the Azure SQL Database and Stream Analytics experiences.



## Key benefits

- Minimum context switching: You can start from a SQL Database in the portal and start ingesting real-time data into a table without switching to any other service.
- Reduced number of steps: The context of your database and table is used to pre-configure a Stream Analytics job.
- Additional ease of use with preview data: Preview incoming data from the events source (Event Hub/IoT Hub) in the context of selected table

## Prerequisites

To complete the steps in this article, you need the following resources:

- An Azure subscription. If you don't have an Azure subscription, [create a free account](#).
- A SQL database. For details, see [Create a single database in Azure SQL Database](#).
- A firewall rule allowing your computer to connect to the Azure SQL server. For details, see [Create a server-level firewall rule](#).

## Configure Stream analytics integration

1. Sign in to the Azure portal.
2. Navigate to your SQL database where you want to ingest your streaming data. Select **Stream analytics**

(preview).

The screenshot shows the Stream Analytics (preview) interface. On the left, there's a navigation sidebar with options like Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), PowerApps (preview), Settings (Configure, Geo-Replication, Connection strings, Sync to other databases, Add Azure Search, Properties, Locks, Export template), Integrations (Stream analytics (preview) - highlighted with a red box), Security (Advanced data security, Auditing, Dynamic Data Masking, Transparent data encryption), and Intelligent Performance. The main area is titled 'Stream Analytics jobs' with a search bar. It features a large gear icon and the message 'No jobs to display'. Below this, it says 'SQL database has integrated with Azure Stream Analytics to allow you to perform near real-time analytics on streaming data. Create a job right from your database.' A 'Create' button is at the bottom. The URL in the browser is 'Home > Microsoft.SQLDatabase.newDatabaseNewServer\_6a832873258e451491d74 - Overview > contososqldb (contososqlserver&lt;/&gt;contososqldb) - Stream analytics (preview)'.

3. To start ingesting your streaming data into this SQL database, select **Create** and give a name to your streaming job, and then select **Next: Input**.

This screenshot shows the 'Create Stream Analytics job' wizard. The left side is the same Stream Analytics interface as the previous screenshot. The right side is the 'Create Stream Analytics job' dialog. It has tabs for Basics, Input, and Output, with Basics selected. The 'Job name \*' field contains 'contoso-job'. At the bottom, there are 'Create', 'Next: Input >', and '< Previous' buttons. The URL in the browser is 'Home > Microsoft.SQLDatabase.newDatabaseNewServer\_6a832873258e451491d74 - Overview > contososqldb (contososqlserver&lt;/&gt;contososqldb) - Stream analytics (preview)'.

4. Enter your events source details, and then select **Next: Output**.

- **Input type:** Event Hub/IoT Hub
- **Input alias:** Enter a name to identify your events source
- **Subscription:** Same as SQL Database subscription
- **Event Hub namespace:** Name for namespace
- **Event Hub name:** Name of event hub within selected namespace
- **Event Hub policy name** (Default to create new): Give a policy name
- **Event Hub consumer group** (Default to create new): Give a consumer group name
  - We recommend that you create a consumer group and a policy for each new Azure Stream Analytics job that you create from here. Consumer groups allow only five concurrent readers, so providing a dedicated consumer group for each job will avoid any errors that might arise from exceeding that limit. A dedicated policy allows you to rotate your key or revoke permissions without impacting other resources.

5. Select which table you want to ingest your streaming data into. Once done, select **Create**.

- **Username, Password:** Enter your credentials for SQL server authentication. Select **Validate**.
- **Table:** Select **Create new** or **Use existing**. In this flow, let's select **Create**. This will create a new table when you start the stream Analytics job.

6. A query page opens with following details:

- Your **Input** (input events source) from which you will ingest data
- Your **Output** (output table) which will store transformed data
- Sample [SAQL query](#) with SELECT statement.
- **Input preview:** Shows snapshot of latest incoming data from input events source.
  - The serialization type in your data is automatically detected (JSON/CSV). You can manually change it as well to JSON/CSV/AVRO.
  - You can preview incoming data in the Table format or Raw format.
  - If your data shown isn't current, select **Refresh** to see the latest events.
  - Select **Select time range** to test your query against a specific time range of incoming events.
  - Select **Upload sample input** to test your query by uploading a sample JSON/CSV file. For more information about testing a SAQL query, see [Test an Azure Stream Analytics job with sample data](#).

The screenshot shows the Azure Stream Analytics job interface. In the top navigation bar, the path is: Home > Microsoft.SQLDatabase.newDatabaseViewServer\_6a832873258e451491d74 - Overview > contososqlDb (contososqlserver/contososqlDb) - Stream analytics (preview) > contoso-job. The main area displays a Stream Analytics job named "contoso-job". It has one input, "contoso-eventhub", and one output, "contososqlDb". The query editor contains a "Test query" section with the following SQL:

```

1 Test query
2
3 INTO
4 [contososqlDb]
5 FROM
6 [contoso-eventhub]

```

Below the query editor, there are tabs for "Input preview", "Test results", "Test results schema", and "Output schema". The "Test results" tab is selected, showing a table of event data from "contoso-eventhub". The table includes columns: from, dspl, time, temp, hmdt, lght, EventProcessedUtcTime, PartitionId, and EventEnqueuedUtcTime. The data rows show various sensor readings over time.

- **Test results:** Select **Test query** and you can see the results of your streaming query

This screenshot shows the same Azure Stream Analytics job interface as the previous one, but with a different query in the editor:

```

1 SELECT
2 dspl as Room, AVG(temp) as Temperature, AVG(hmdt) as Humidity, System.Timestamp as [Timestamp]
3 INTO
4 [contososqlDb]
5 FROM
6 [contoso-eventhub] TIMESTAMP BY time
7 GROUP BY dspl, TUMBLINGWINDOW(second, 5)

```

The "Test results" tab is highlighted with a red box. The results table shows 8 rows from "contososqlDb" with columns: room, temperature, humidity, and timestamp. The data rows show average values for each room over a 5-second window.

- **Test results schema:** Shows the schema of the results of your streaming query after testing. Make sure the test results schema matches with your output schema.

This screenshot shows the same Azure Stream Analytics job interface, but with the "Test results schema" tab selected. The query in the editor remains the same as the previous screenshots:

```

1 SELECT
2 dspl as Room, AVG(temp) as Temperature, AVG(hmdt) as Humidity, System.Timestamp as [Timestamp]
3 INTO
4 [contososqlDb]
5 FROM
6 [contoso-eventhub] TIMESTAMP BY time
7 GROUP BY dspl, TUMBLINGWINDOW(second, 5)

```

The "Test results schema" tab is highlighted with a red box. The results table shows the schema of the output table "contososqlDb" based on the query. It includes columns: room (string), temperature (float), humidity (float), and timestamp (datetime). The "Index" dropdown is set to "NONCLUSTERED".

- **Output schema:** This contains schema of the table you selected in step 5 (new or existing).

- Create new: If you selected this option in step 5, you won't see the schema yet until you start the streaming job. When creating a new table, select the appropriate table index. For more

information about table indexing, see [Clustered and Nonclustered Indexes Described](#).

- Use existing: If you selected this option in step 5, you will see the schema of selected table.
7. After you are done authoring & testing the query, select **Save query**. Select **Start Stream Analytics job** to start ingesting transformed data into the SQL table. Once you finalize the following fields, **start** the job.

- **Output start time:** This defines the time of the first output of the job.
  - Now: The job will start now and process new incoming data.
  - Custom: The job will start now but will process data from a specific point in time (that can be in the past or the future). For more information, see [How to start an Azure Stream Analytics job](#).
- **Streaming units:** Azure Stream Analytics is priced by the number of streaming units required to process the data into the service. For more information, see [Azure Stream Analytics pricing](#).
- **Output data error handling:**
  - Retry: When an error occurs, Azure Stream Analytics retries writing the event indefinitely until the write succeeds. There is no timeout for retries. Eventually all subsequent events are blocked from processing by the event that is retrying. This option is the default output error handling policy.
  - Drop: Azure Stream Analytics will drop any output event that results in a data conversion error. The dropped events cannot be recovered for reprocessing later. All transient errors (for example, network errors) are retried regardless of the output error handling policy configuration.
- **SQL Database output settings:** An option for inheriting the partitioning scheme of your previous query step, to enable fully parallel topology with multiple writers to the table. For more information, see [Azure Stream Analytics output to Azure SQL Database](#).
- **Max batch count:** The recommended upper limit on the number of records sent with every bulk insert transaction.

For more information about output error handling, see [Output error policies in Azure Stream Analytics](#).

The screenshot shows the Azure Stream Analytics job configuration interface. On the left, there's a navigation bar with 'Home', 'contososqldb (contososeqlserver8/contososqldb)', 'Stream analytics (preview)', and 'contoso-job'. Below this are sections for 'Inputs (1)' (contoso-eventhub) and 'Outputs (1)' (contososqldb). The main area contains a SQL query editor with the following code:

```
1 SELECT
2 dspl as Room, AVG(temp) as Temperature, AVG(hmdt) as Humidity, System.Timestamp as [Timestamp]
3 INTO
4 [contososqldb]
5 FROM
6 [contoso-eventhub] TIMESTAMP BY time
7 GROUP BY dspl, TUMBLINGWINDOW(second, 5)
```

Below the query editor, there are tabs for 'Input preview', 'Test results', 'Test results schema', and 'Output schema'. The 'Input preview' tab shows 11 rows of data:

room	temperature	humidity
"Conf Room 34/1301 (12) AV"	50	30
"Conf Room 34/1561 (8) AV RoundTable"	55.6	32.8
"Conf Room 34/1301 (12) AV"	49.6	30.4
"Conf Room 34/1561 (8) AV RoundTable"	55.6	32.6
"Conf Room 34/1301 (12) AV"	49.8	30
"Conf Room 34/1561 (8) AV RoundTable"	56	32.8
"Conf Room 34/1301 (12) AV"	49.6	29.8
"Conf Room 34/1561 (8) AV RoundTable"	55.6	33
"Conf Room 34/1301 (12) AV"	50.5	28.5
"Conf Room 34/1561 (8) AV RoundTable"	55	32.4
"Conf Room 34/1561 (8) AV RoundTable"	56.333333333333336	31.666666666666668

On the right, a 'Start Stream Analytics job' dialog box is open. It includes fields for 'Start this Stream Analytics job to ingest streaming data from Event Hub IoT Hub into a SQL table.', 'Output start time' (radio buttons for 'Now' and 'Custom'), 'Streaming units' (set to 6), 'Output data error handling' (radio buttons for 'Retry' and 'Drop'), and 'SQL Database output settings' (radio buttons for 'Merge all input partitions into a single writer' and 'Inherit partition scheme of previous query step or input'). A 'Max batch count' field is set to 10000. At the bottom right of the dialog is a large red-bordered 'Start' button.

8. Once you start the job, you will see the Running job in the list, and you can take following actions:

- **Start/stop the job:** If the job is running, you can stop the job. If the job is stopped, you can start the job.
- **Edit job:** You can edit the query. If you want to do more changes to the job ex, add more inputs/outputs, then open the job in Stream Analytics. Edit option is disabled when the job is running.
- **Preview output table:** You can preview the table in SQL query editor.
- **Open in Stream Analytics:** Open the job in Stream Analytics service to view monitoring, debugging details of the job.

The screenshot shows the Stream Analytics Jobs page in the Azure portal. A single job named "contoso-job" is listed. The job has "contoso-eventhub" as its input and "contososqlDb" as its output. It is located in "East US" and is currently "Running". A context menu is open over the "Running" status, with options including "Stop job", "Edit job", "Preview output table", and "Open in Stream Analytics".

Name	Input	Output	Location	Status
contoso-job	contoso-eventhub	contososqlDb	East US	Running

## Next steps

- [Azure Stream Analytics documentation](#)
- [Azure Stream Analytics solution patterns](#)

# How to migrate your SQLite database to Azure SQL Database Serverless

1/9/2020 • 3 minutes to read • [Edit Online](#)

For many people, SQLite provides their first experience of databases and SQL programming. Its inclusion in many operating systems and popular applications makes SQLite one of the most widely deployed and used database engines in the world. And because it is likely the first database engine many people use, it can often end up as a central part of projects or applications. In such cases where the project or application outgrows the initial SQLite implementation, developers may need to migrate their data to a reliable, centralized data store.

Azure SQL Database serverless is a compute tier for single databases that automatically scales compute based on workload demand, and bills for the amount of compute used per second. The serverless compute tier also automatically pauses databases during inactive periods when only storage is billed and automatically resumes databases when activity returns.

Once you have followed the below steps, your database will be migrated into Azure SQL Database Serverless, enabling you to make your database available to other users or applications in the cloud and only pay for what you use, with minimal application code changes.

## Prerequisites

- An Azure Subscription
- SQLite2 or SQLite3 database that you wish to migrate
- A Windows environment
  - If you do not have a local Windows environment, you can use a Windows VM in Azure for the migration.  
Move and make your SQLite database file available on the VM using Azure Files and Storage Explorer.

## Steps

1. Provision a new Azure SQL Database in the Serverless compute tier.

## Configure

 Feedback

 Looking for basic, standard, premium?

### General Purpose

Scalable compute and storage options

500 - 20,000 IOPS  
2-10 ms latency

### Hyperscale

On-demand scalable storage

500 - 204,800 IOPS  
1-10 ms latency

### Compute tier

#### Provisioned

Compute resources are pre-allocated  
Billed per hour based on vCores configured

#### Serverless

Compute resources are auto-scaled  
Billed per second based on vCores used

### Compute Hardware

Click "Change configuration" to see details for all hardware generations available including memory optimized and compute optimized options

#### Hardware Configuration

##### Gen5

up to 16 vCores, up to 48 GB memory  
[Change configuration](#)

2. Ensure you have your SQLite database file available in your Windows environment. Install a SQLite ODBC Driver if you do not already have one (there are many available in Open Source, for example, <http://www.ch-werner.de/sqliteodbc/>).
3. Create a System DSN for the database. Ensure you use the Data Source Administrator application that matches your system architecture (32-bit vs 64-bit). You can find which version you are running in your system settings.
  - Open ODBC Data Source Administrator in your environment.
  - Click the system DSN tab and click "Add"
  - Select the SQLite ODBC connector you installed and give the connection a meaningful name, for example, sqitemigrationsource
  - Set the database name to the .db file
  - Save and exit
4. Download and install the self-hosted integration runtime. The easiest way to do this is the Express install option, as detailed in the documentation. If you opt for a manual install, you will need to provide the application with an authentication key, which can be located in your Data Factory instance by:
  - Starting up ADF (Author and Monitor from the service in the Azure portal)
  - Click the "Author" tab (Blue pencil) on the left
  - Click Connections (bottom left), then Integration runtimes
  - Add new Self-Hosted Integration Runtime, give it a name, select *Option 2*.
5. Create a new linked service for the source SQLite database in your Data Factory.

The screenshot shows the Microsoft Azure Data Factory interface. The left sidebar lists 'Factory Resources' with options for Pipelines (0), Datasets (0), and Data flows (0). The main area is titled 'Connections' and shows the 'Linked services' tab selected. A search bar at the top right says 'Search resources'. Below the tabs, there's a button '+ New' and a search bar 'Search to filter items...'. The table below has columns for NAME, TYPE, and ANNOTATIONS. The bottom navigation bar has tabs for 'Connections' (selected) and 'Triggers'.

6. In Connections, under Linked Service, click New
7. Search for and select the "ODBC" connector

## New linked service

Data store   Compute

 odbc

All   Azure   Database   File   Generic protocol   NoSQL   Services and apps



ODBC

Continue

Cancel

8. Give the linked service a meaningful name, for example, "sqlite\_odbc". Select your integration runtime from the "Connect via integration runtime" dropdown. Enter the below into the connection string, replacing the Initial Catalog variable with the filepath for the .db file, and the DSN with the name of the system DSN connection:

```
Connection string: Provider=MSDASQL.1;Persist Security Info=False;Mode=ReadWrite;Initial Catalog=C:\sqlitemigrationsource.db;DSN=sqlitemigrationsource
```

9. Set the authentication type to Anonymous

10. Test the connection

## New linked service (ODBC)

Name \*

Description

Connect via integration runtime \*

sqlitemigration-ir

[Edit integration runtime](#)

Connection string	Azure Key Vault
Connection string *	
Connection string: Provider=MSDASQL.1;Persist Security Info=False;Mode=ReadWrite;Initial Catalog=C:\sqlite\sqlitemigrationsource;DSN=sqlitemigrationsource	

Authentication type \*

Anonymous

Credential	Azure Key Vault
Credential	

✓ Connection successful

[Create](#) [Back](#) [!\[\]\(feb93e0bbe8f0bf5997e5a2fae8b0fd8\_img.jpg\) Test connection](#) [Cancel](#)

11. Create another linked service for your Serverless SQL target. Select the database using the linked service wizard, and provide the SQL authentication credentials.

## New linked service

**Data store**   [Compute](#)

All   [Azure](#)   **Database**   File   Generic protocol   NoSQL   Services and apps

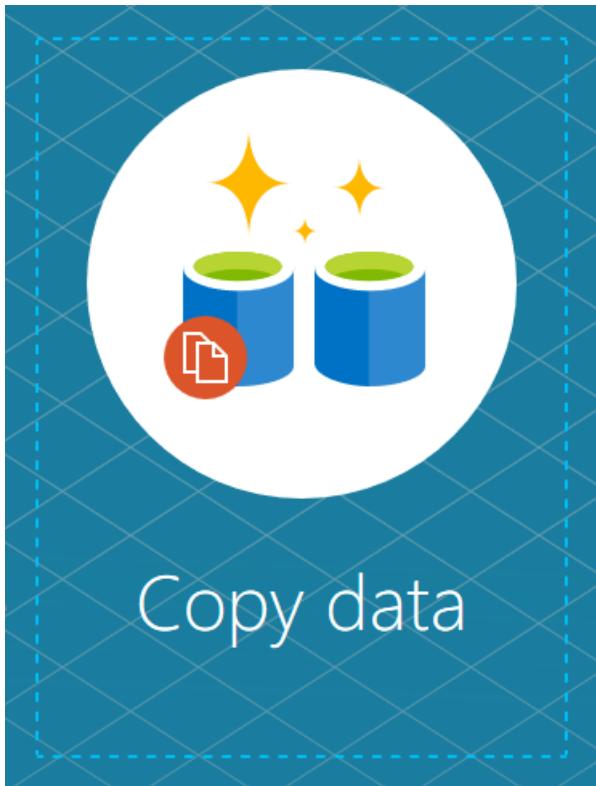
 Azure Cosmos DB (SQL API)	 Azure Database for MySQL	 Azure Database for PostgreSQL
 Azure SQL Database	 Azure SQL Database Managed Instance	 Azure Synapse Analytics (formerly SQL DW)
		
<a href="#">Continue</a>		<a href="#">Cancel</a>

12. Extract the CREATE TABLE statements from your SQLite database. You can do this by executing the below Python script on your database file.

```
#!/usr/bin/python
import sqlite3
conn = sqlite3.connect("sqlitemigrationsource.db")
c = conn.cursor()

print("Starting extract job..")
with open('CreateTables.sql', 'w') as f:
 for tabledetails in c.execute("SELECT * FROM sqlite_master WHERE type='table'"):
 print("Extracting CREATE statement for " + (str(tabledetails[1])))
 print(tabledetails)
 f.write(str(tabledetails[4].replace('\n','')) + '\n')
c.close()
```

13. Create the landing tables in your Serverless SQL target environment by copying the CREATE table statements from the CreateTables.sql file and running the SQL statements in the Query Editor in the Azure portal.
14. Return to the home screen of your Data Factory and click "Copy Data" to run through the job creation wizard.



## Copy data

15. Select all tables from the source SQLite database using the check boxes, and map them to the target tables in Azure SQL. Once the job has run, you have successfully migrated your data from SQLite to Azure SQL!

## Next steps

- To get started, see [Quickstart: Create a single database in Azure SQL Database using the Azure portal](#).
- For resource limits, see [Serverless compute tier resource limits](#).

# Resource limits for single databases using the vCore purchasing model

1/22/2020 • 20 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database single databases using the vCore purchasing model.

For DTU purchasing model limits for single databases on a SQL Database server, see [Overview of resource limits on a SQL Database server](#).

You can set the service tier, compute size, and storage amount for a single database using the [Azure portal](#), [Transact-SQL](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

## IMPORTANT

For scaling guidance and considerations, see [Scale a single database](#).

## General purpose - serverless compute - Gen5

The [serverless compute tier](#) is currently available on Gen5 hardware only.

### Gen5 compute generation (part 1)

COMPUTE SIZE	GP_S_GEN5_1	GP_S_GEN5_2	GP_S_GEN5_4	GP_S_GEN5_6	GP_S_GEN5_8
Compute generation	Gen5	Gen5	Gen5	Gen5	Gen5
Min-max vCores	0.5-1	0.5-2	0.5-4	0.75-6	1.0-8
Min-max memory (GB)	2.02-3	2.05-6	2.10-12	2.25-18	3.00-24
Min auto-pause delay (minutes)	60	60	60	60	60
Columnstore support	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A
Max data size (GB)	512	1024	1024	1024	1536
Max log size (GB)	154	307	307	307	461

<b>COMPUTE SIZE</b>	<b>GP_S_GEN5_1</b>	<b>GP_S_GEN5_2</b>	<b>GP_S_GEN5_4</b>	<b>GP_S_GEN5_6</b>	<b>GP_S_GEN5_8</b>
TempDB max data size (GB)	32	64	128	192	256
Storage type	Remote SSD				
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)				
Max data IOPS *	320	640	1280	1920	2560
Max log rate (MBps)	3.8	7.5	15	22.5	30
Max concurrent workers (requests)	75	150	300	450	600
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000
Number of replicas	1	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A	N/A
Included backup storage	1X DB size				

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.  
For details, see [Data IO Governance](#).

### Gen5 compute generation (part 2)

<b>COMPUTE SIZE</b>	<b>GP_S_GEN5_10</b>	<b>GP_S_GEN5_12</b>	<b>GP_S_GEN5_14</b>	<b>GP_S_GEN5_16</b>
Compute generation	Gen5	Gen5	Gen5	Gen5
Min-max vCores	1.25-10	1.50-12	1.75-14	2.00-16
Min-max memory (GB)	3.75-30	4.50-36	5.25-42	6.00-48
Min auto-pause delay (minutes)	60	60	60	60
Columnstore support	Yes	Yes	Yes	Yes

COMPUTE SIZE	GP_S_GEN5_10	GP_S_GEN5_12	GP_S_GEN5_14	GP_S_GEN5_16
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A
Max data size (GB)	1536	3072	3072	3072
Max log size (GB)	461	461	461	922
TempDB max data size (GB)	320	384	448	512
Storage type	Remote SSD	Remote SSD	Remote SSD	Remote SSD
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)			
Max data IOPS *	3200	3840	4480	5120
Max log rate (MBps)	30	30	30	30
Max concurrent workers (requests)	750	900	1050	1200
Max concurrent sessions	30,000	30,000	30,000	30,000
Number of replicas	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.  
For details, see [Data IO Governance](#).

## Hyperscale - provisioned compute - Gen4

### Gen4 compute generation (part 1)

PERFORMANCE LEVEL	HS_GEN4_1	HS_GEN4_2	HS_GEN4_3	HS_GEN4_4	HS_GEN4_5	HS_GEN4_6
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	1	2	3	4	5	6
Memory (GB)	7	14	21	28	35	42

PERFORMANCE LEVEL	HS_GEN4_1	HS_GEN4_2	HS_GEN4_3	HS_GEN4_4	HS_GEN4_5	HS_GEN4_6
RBPEX Size	3X Memory					
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A	N/A
Max data size (TB)	100	100	100	100	100	100
Max log size (TB)	1	1	1	1	1	1
TempDB max data size (GB)	32	64	96	128	160	192
Storage type	<a href="#">Note 1</a>					
Max data IOPS *	<a href="#">Note 2</a>					
IO latency (approximate )	<a href="#">Note 3</a>					
Max concurrent workers (requests)	200	400	600	800	1000	1200
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Secondary replicas	0-4	0-4	0-4	0-4	0-4	0-4
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes
Backup storage retention	7 days					

## Gen4 compute generation (part 2)

PERFORMANCE LEVEL	HS_GEN4_7	HS_GEN4_8	HS_GEN4_9	HS_GEN4_10	HS_GEN4_16	HS_GEN4_24
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	7	8	9	10	16	24
Memory (GB)	49	56	63	70	112	159.5
RBPEX Size	3X Memory					
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A	N/A
Max data size (TB)	100	100	100	100	100	100
Max log size (TB)	1	1	1	1	1	1
TempDB max data size (GB)	224	256	288	320	512	768
Storage type	<a href="#">Note 1</a>					
Max data IOPS *	<a href="#">Note 2</a>					
IO latency (approximate )	<a href="#">Note 3</a>					
Max concurrent workers (requests)	1400	1600	1800	2000	3200	4800
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Secondary replicas	0-4	0-4	0-4	0-4	0-4	0-4
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes

PERFORMANCE LEVEL	HS_GEN4_7	HS_GEN4_8	HS_GEN4_9	HS_GEN4_10	HS_GEN4_16	HS_GEN4_24
Backup storage retention	7 days	7 days	7 days	7 days	7 days	7 days

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

# Hyperscale - provisioned compute - Gen5

## Gen5 compute generation (part 1)

PERFORMANCE LEVEL	HS_GEN5_2	HS_GEN5_4	HS_GEN5_6	HS_GEN_8	HS_GEN5_10	HS_GEN5_12	HS_GEN5_14
IO latency (approximate)	Note 3	Note 3	Note 3	Note 3	Note 3	Note 3	Note 3
Max concurrent workers (requests)	200	400	600	800	1000	1200	1400
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Secondary replicas	0-4	0-4	0-4	0-4	0-4	0-4	0-4
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Backup storage retention	7 days	7 days	7 days	7 days	7 days	7 days	7 days

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## Gen5 compute generation (part 2)

PERFORMANCE LEVEL	HS_GEN5_16	HS_GEN5_18	HS_GEN5_20	HS_GEN_24	HS_GEN5_32	HS_GEN5_40	HS_GEN5_80
Compute generation	Gen5	Gen5	Gen5	Gen5	Gen5	Gen5	Gen5
vCores	16	18	20	24	32	40	80
Memory (GB)	83	93.4	103.8	124.6	166.1	207.6	415.2
RBPEX Size	3X Memory	3X Memory	3X Memory	3X Memory	3X Memory	3X Memory	3X Memory
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A	N/A	N/A

PERFORMANCE LEVEL	HS_GEN5_16	HS_GEN5_18	HS_GEN5_20	HS_GEN24	HS_GEN5_32	HS_GEN5_40	HS_GEN5_80
Max data size (TB)	100	100	100	100	100	100	100
Max log size (TB)	1	1	1	1	1	1	1
TempDB max data size (GB)	512	576	640	768	1024	1280	2560
Storage type	Note 1	Note 1	Note 1	Note 1	Note 1	Note 1	Note 1
Max data IOPS *	Note 2	Note 2	Note 2	Note 2	Note 2	Note 2	Note 2
IO latency (approximate)	Note 3	Note 3	Note 3	Note 3	Note 3	Note 3	Note 3
Max concurrent workers (requests)	1600	1800	2000	2400	3200	4000	8000
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Secondary replicas	0-4	0-4	0-4	0-4	0-4	0-4	0-4
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Backup storage retention	7 days	7 days	7 days	7 days	7 days	7 days	7 days

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.

For details, see [Data IO Governance](#).

#### Notes

**Note 1:** Hyperscale is a multi-tiered architecture with separate compute and storage components:

[Hyperscale Service Tier Architecture](#)

**Note 2:** Hyperscale multi-tiered architecture has caching at multiple levels. Effective IOPS will depend on the workload.

**Note 3:** Latency is 1-2 ms for data in the RBPEX SSD-based cache on compute replicas, which caches most used data pages. Higher latency for data retrieved from page servers.

# General purpose - provisioned compute - Gen4

## IMPORTANT

New Gen4 databases are no longer supported in the Australia East or Brazil South regions.

## Gen4 compute generation (part 1)

COMPUTE SIZE	GP_GEN4_1	GP_GEN4_2	GP_GEN4_3	GP_GEN4_4	GP_GEN4_5	GP_GEN4_6
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	1	2	3	4	5	6
Memory (GB)	7	14	21	28	35	42
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A	N/A
Max data size (GB)	1024	1024	1536	1536	1536	3072
Max log size (GB)	307	307	461	461	461	922
TempDB max data size (GB)	32	64	96	128	160	192
Storage type	Remote SSD					
IO latency (approximate )	5-7 ms (write) 5-10 ms (read)					
Max data IOPS *	320	640	960	1280	1600	1920
Max log rate (MBps)	3.75	7.5	11.25	15	18.75	22.5
Max concurrent workers (requests)	200	400	600	800	1000	1200

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## Gen4 compute generation (part 2)

<b>COMPUTE SIZE</b>	<b>GP_GEN4_7</b>	<b>GP_GEN4_8</b>	<b>GP_GEN4_9</b>	<b>GP_GEN4_10</b>	<b>GP_GEN4_16</b>	<b>GP_GEN4_24</b>
IO latency (approximate )	5-7 ms (write) 5-10 ms (read)					
Max data IOPS *	2240	2560	2880	3200	5120	7680
Max log rate (MBps)	26.3	30	30	30	30	30
Max concurrent workers (requests)	1400	1600	1800	2000	3200	4800
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Number of replicas	1	1	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A	N/A	N/A
Included backup storage	1X DB size					

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## General purpose - provisioned compute - Gen5

### Gen5 compute generation (part 1)

<b>COMPUTE SIZE</b>	<b>GP_GEN5_2</b>	<b>GP_GEN5_4</b>	<b>GP_GEN5_6</b>	<b>GP_GEN5_8</b>	<b>GP_GEN5_10</b>	<b>GP_GEN5_12</b>	<b>GP_GEN5_14</b>
Compute generation	Gen5	Gen5	Gen5	Gen5	Gen5	Gen5	Gen5
vCores	2	4	6	8	10	12	14
Memory (GB)	10.4	20.8	31.1	41.5	51.9	62.3	72.7
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes	Yes

<b>COMPUTE SIZE</b>	<b>GP_GEN5_2</b>	<b>GP_GEN5_4</b>	<b>GP_GEN5_6</b>	<b>GP_GEN5_8</b>	<b>GP_GEN5_10</b>	<b>GP_GEN5_12</b>	<b>GP_GEN5_14</b>
In-memory OLTP storage (GB)	N/A						
Max data size (GB)	1024	1024	1536	1536	1536	3072	3072
Max log size (GB)	307	307	461	461	461	922	922
TempDB max data size (GB)	64	128	192	256	320	384	384
Storage type	Remote SSD						
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)						
Max data IOPS *	640	1280	1920	2560	3200	3840	4480
Max log rate (MBps)	7.5	15	22.5	30	30	30	30
Max concurrent workers (requests)	200	400	600	800	1000	1200	1400
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Number of replicas	1	1	1	1	1	1	1
Multi-AZ	N/A						
Read Scale-out	N/A						
Included backup storage	1X DB size						

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## Gen5 compute generation (part 2)

<b>COMPUTE SIZE</b>	<b>GP_GEN5_16</b>	<b>GP_GEN5_18</b>	<b>GP_GEN5_20</b>	<b>GP_GEN5_24</b>	<b>GP_GEN5_32</b>	<b>GP_GEN5_40</b>	<b>GP_GEN5_80</b>
Compute generation	Gen5						
vCores	16	18	20	24	32	40	80
Memory (GB)	83	93.4	103.8	124.6	166.1	207.6	415.2
Columns to re support	Yes						
In-memory OLTP storage (GB)	N/A						
Max data size (GB)	3072	3072	3072	4096	4096	4096	4096
Max log size (GB)	922	922	922	1229	1229	1229	1229
TempDB max data size (GB)	512	576	640	768	1024	1280	2560
Storage type	Remote SSD						
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)						
Max data IOPS *	5120	5760	6400	7680	10240	12800	25600
Max log rate (MBps)	30	30	30	30	30	30	30
Max concurrent workers (requests)	1600	1800	2000	2400	3200	4000	8000
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Number of replicas	1	1	1	1	1	1	1

<b>COMPUTE SIZE</b>	<b>GP_GEN5_16</b>	<b>GP_GEN5_18</b>	<b>GP_GEN5_20</b>	<b>GP_GEN5_24</b>	<b>GP_GEN5_32</b>	<b>GP_GEN5_40</b>	<b>GP_GEN5_80</b>
Multi-AZ	N/A						
Read Scale-out	N/A						
Included backup storage	1X DB size						

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## General purpose - provisioned compute - Fsv2-series

### Fsv2-series compute generation (preview)

<b>COMPUTE SIZE</b>	<b>GP_FSV2_72</b>
Compute generation	Fsv2-series
vCores	72
Memory (GB)	136.2
Columnstore support	Yes
In-memory OLTP storage (GB)	N/A
Max data size (GB)	4096
Max log size (GB)	1024
TempDB max data size (GB)	333
Storage type	Remote SSD
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)
Max data IOPS *	12,800
Max log rate (MBps)	30
Max concurrent workers (requests)	3600
Max concurrent logins	3600
Max concurrent sessions	30,000
Number of replicas	1

Compute Size	GP_FSV2_72
Multi-AZ	N/A
Read Scale-out	N/A
Included backup storage	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## Business critical - provisioned compute - Gen4

## **IMPORTANT**

New Gen4 databases are no longer supported in the Australia East or Brazil South regions.

## Gen4 compute generation (part 1)

<b>COMPUTE SIZE</b>	<b>BC_GEN4_1</b>	<b>BC_GEN4_2</b>	<b>BC_GEN4_3</b>	<b>BC_GEN4_4</b>	<b>BC_GEN4_5</b>	<b>BC_GEN4_6</b>
Max data IOPS *	4,000	8,000	12,000	16,000	20,000	24,000
Max log rate (MBps)	8	16	24	32	40	48
Max concurrent workers (requests)	200	400	600	800	1000	1200
Max concurrent logins	200	400	600	800	1000	1200
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Number of replicas	4	4	4	4	4	4
Multi-AZ	Yes	Yes	Yes	Yes	Yes	Yes
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes
Included backup storage	1X DB size					

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.

For details, see [Data IO Governance](#).

### Gen4 compute generation (part 2)

<b>COMPUTE SIZE</b>	<b>BC_GEN4_7</b>	<b>BC_GEN4_8</b>	<b>BC_GEN4_9</b>	<b>BC_GEN4_10</b>	<b>BC_GEN4_16</b>	<b>BC_GEN4_24</b>
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	7	8	9	10	16	24
Memory (GB)	49	56	63	70	112	159.5
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	7	8	9.5	11	20	36

<b>COMPUTE SIZE</b>	<b>BC_GEN4_7</b>	<b>BC_GEN4_8</b>	<b>BC_GEN4_9</b>	<b>BC_GEN4_10</b>	<b>BC_GEN4_16</b>	<b>BC_GEN4_24</b>
Storage type	Local SSD					
Max data size (GB)	1024	1024	1024	1024	1024	1024
Max log size (GB)	307	307	307	307	307	307
TempDB max data size (GB)	224	256	288	320	512	768
IO latency (approximate )	1-2 ms (write) 1-2 ms (read)					
Max data IOPS	28,000	32,000	36,000	40,000	64,000	76,800
Max log rate (MBps)	56	64	64	64	64	64
Max concurrent workers (requests)	1400	1600	1800	2000	3200	4800
Max concurrent logins (requests)	1400	1600	1800	2000	3200	4800
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Number of replicas	4	4	4	4	4	4
Multi-AZ	Yes	Yes	Yes	Yes	Yes	Yes
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes
Included backup storage	1X DB size					

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.  
For details, see [Data IO Governance](#).

## Business critical - provisioned compute - Gen5

## Gen5 compute generation (part 1)

COMPUTE SIZE	BC_GEN5_2	BC_GEN5_4	BC_GEN5_6	BC_GEN5_8	BC_GEN5_10	BC_GEN5_12	BC_GEN5_14
Compute generation	Gen5						
vCores	2	4	6	8	10	12	14
Memory (GB)	10.4	20.8	31.1	41.5	51.9	62.3	72.7
Columnstore support	Yes						
In-memory OLTP storage (GB)	1.57	3.14	4.71	6.28	8.65	11.02	13.39
Max data size (GB)	1024	1024	1536	1536	1536	3072	3072
Max log size (GB)	307	307	461	461	461	922	922
TempDB max data size (GB)	64	128	192	256	320	384	448
Storage type	Local SSD						
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)						
Max data IOPS *	8000	16,000	24,000	32,000	40,000	48,000	56,000
Max log rate (MBps)	24	48	72	96	96	96	96
Max concurrent workers (requests)	200	400	600	800	1000	1200	1400
Max concurrent logins	200	400	600	800	1000	1200	1400

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## Gen5 compute generation (part 2)

<b>COMPUTE SIZE</b>	<b>BC_GEN5_16</b>	<b>BC_GEN5_18</b>	<b>BC_GEN5_20</b>	<b>BC_GEN5_24</b>	<b>BC_GEN5_32</b>	<b>BC_GEN5_40</b>	<b>BC_GEN5_80</b>
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)						
Max data IOPS *	64,000	72,000	80,000	96,000	128,000	160,000	204,800
Max log rate (MBps)	96	96	96	96	96	96	96
Max concurrent workers (requests)	1600	1800	2000	2400	3200	4000	8000
Max concurrent logins	1600	1800	2000	2400	3200	4000	8000
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Number of replicas	4	4	4	4	4	4	4
Multi-AZ	Yes						
Read Scale-out	Yes						
Included backup storage	1X DB size						

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

## Business critical - provisioned compute - M-series

### M-series compute generation (preview)

<b>COMPUTE SIZE</b>	<b>BC_M_128</b>
Compute generation	M-series
vCores	128
Memory (GB)	3767.1
Columnstore support	Yes

COMPUTE SIZE	BC_M_128
In-memory OLTP storage (GB)	1768
Max data size (GB)	4096
Max log size (GB)	2048
TempDB max data size (GB)	4096
Storage type	Local SSD
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)
Max data IOPS *	160,000
Max log rate (MBps)	264
Max concurrent workers (requests)	12,800
Max concurrent logins	12,800
Max concurrent sessions	30000
Number of replicas	4
Multi-AZ	Yes
Read Scale-out	Yes
Included backup storage	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.  
For details, see [Data IO Governance](#).

#### IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

## Next steps

- For DTU resource limits for a single database, see [resource limits for single databases using the DTU purchasing model](#)
- For vCore resource limits for elastic pools, see [resource limits for elastic pools using the vCore purchasing model](#)
- For DTU resource limits for elastic pools, see [resource limits for elastic pools using the DTU purchasing model](#)
- For resource limits for managed instances, see [managed instance resource limits](#).
- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).

- For information about resource limits on a database server, see [overview of resource limits on a SQL Database server](#) for information about limits at the server and subscription levels.

# Resource limits for single databases using the DTU purchasing model

1/3/2020 • 3 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database single databases using the DTU purchasing model.

For DTU purchasing model resource limits for elastic pools, see [DTU resource limits - elastic pools](#). For vCore resource limits, see [vCore resource limits - single databases](#) and [vCore resource limits - elastic pools](#). For more information regarding the different purchasing models, see [Purchasing models and service tiers](#).

## Single database: Storage sizes and compute sizes

The following tables show the resources available for a single database at each service tier and compute size. You can set the service tier, compute size, and storage amount for a single database using the [Azure portal](#), [Transact-SQL](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

### IMPORTANT

For scaling guidance and considerations, see [Scale a single database](#)

### Basic service tier

COMPUTE SIZE	BASIC
Max DTUs	5
Included storage (GB)	2
Max storage choices (GB)	2
Max in-memory OLTP storage (GB)	N/A
Max concurrent workers (requests)	30
Max concurrent sessions	300

### IMPORTANT

The Basic service tier provides less than one vCore (CPU). For CPU-intensive workloads, a service tier of S3 or greater is recommended.

Regarding data storage, the Basic service tier is placed on Standard Page Blobs. Standard Page Blobs use hard disk drive (HDD)-based storage media and are best suited for development, testing, and other infrequently accessed workloads that are less sensitive to performance variability.

### Standard service tier

<b>COMPUTE SIZE</b>	<b>S0</b>	<b>S1</b>	<b>S2</b>	<b>S3</b>
Max DTUs	10	20	50	100
Included storage (GB)	250	250	250	250
Max storage choices (GB)	250	250	250	250, 500, 750, 1024
Max in-memory OLTP storage (GB)	N/A	N/A	N/A	N/A
Max concurrent workers (requests)	60	90	120	200
Max concurrent sessions	600	900	1200	2400

#### **IMPORTANT**

The Standard S0, S1 and S2 tiers provide less than one vCore (CPU). For CPU-intensive workloads, a service tier of S3 or greater is recommended.

Regarding data storage, the Standard S0 and S1 service tiers are placed on Standard Page Blobs. Standard Page Blobs use hard disk drive (HDD)-based storage media and are best suited for development, testing, and other infrequently accessed workloads that are less sensitive to performance variability.

#### **Standard service tier (continued)**

<b>COMPUTE SIZE</b>	<b>S4</b>	<b>S6</b>	<b>S7</b>	<b>S9</b>	<b>S12</b>
Max DTUs	200	400	800	1600	3000
Included storage (GB)	250	250	250	250	250
Max storage choices (GB)	250, 500, 750, 1024	250, 500, 750, 1024	250, 500, 750, 1024	250, 500, 750, 1024	250, 500, 750, 1024
Max in-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A
Max concurrent workers (requests)	400	800	1600	3200	6000
Max concurrent sessions	4800	9600	19200	30000	30000

#### **Premium service tier**

<b>COMPUTE SIZE</b>	<b>P1</b>	<b>P2</b>	<b>P4</b>	<b>P6</b>	<b>P11</b>	<b>P15</b>
Max DTUs	125	250	500	1000	1750	4000
Included storage (GB)	500	500	500	500	4096*	4096*
Max storage choices (GB)	500, 750, 1024	500, 750, 1024	500, 750, 1024	500, 750, 1024	4096*	4096*
Max in-memory OLTP storage (GB)	1	2	4	8	14	32
Max concurrent workers (requests)	200	400	800	1600	2400	6400
Max concurrent sessions	30000	30000	30000	30000	30000	30000

\* From 1024 GB up to 4096 GB in increments of 256 GB

#### IMPORTANT

More than 1 TB of storage in the Premium tier is currently available in all regions except: China East, China North, Germany Central, Germany Northeast, West Central US, US DoD regions, and US Government Central. In these regions, the storage max in the Premium tier is limited to 1 TB. For more information, see [P11-P15 current limitations](#).

#### NOTE

For `tempdb` limits, see [tempdb limits](#).

## Next steps

- For vCore resource limits for a single database, see [resource limits for single databases using the vCore purchasing model](#)
- For vCore resource limits for elastic pools, see [resource limits for elastic pools using the vCore purchasing model](#)
- For DTU resource limits for elastic pools, see [resource limits for elastic pools using the DTU purchasing model](#)
- For resource limits for managed instances, see [managed instance resource limits](#).
- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).
- For information about resource limits on a database server, see [overview of resource limits on a SQL Database server](#) for information about limits at the server and subscription levels.

# Scale single database resources in Azure SQL Database

2/4/2020 • 9 minutes to read • [Edit Online](#)

This article describes how to scale the compute and storage resources available for an Azure SQL Database in the provisioned compute tier. Alternatively, the [serverless compute tier](#) provides compute auto-scaling and bills per second for compute used.

After initially picking the number of vCores or DTUs, you can scale a single database up or down dynamically based on actual experience using the [Azure portal](#), [Transact-SQL](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

The following video shows dynamically changing the service tier and compute size to increase available DTUs for a single database.

## IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

## Impact

Changing the service tier or compute size of mainly involves the service performing the following steps:

### 1. Create new compute instance for the database

A new compute instance is created with the requested service tier and compute size. For some combinations of service tier and compute size changes, a replica of the database must be created in the new compute instance which involves copying data and can strongly influence the overall latency. Regardless, the database remains online during this step, and connections continue to be directed to the database in the original compute instance.

### 2. Switch routing of connections to new compute instance

Existing connections to the database in the original compute instance are dropped. Any new connections are established to the database in the new compute instance. For some combinations of service tier and compute size changes, database files are detached and reattached during the switch. Regardless, the switch can result in a brief service interruption when the database is unavailable generally for less than 30 seconds and often for only a few seconds. If there are long running transactions running when connections are dropped, the duration of this step may take longer in order to recover aborted transactions. [Accelerated Database Recovery](#) can reduce the impact from aborting long running transactions.

## IMPORTANT

No data is lost during any step in the workflow. Make sure that you have implemented some [retry logic](#) in the applications and components that are using Azure SQL Database while the service tier is changed.

# Latency

The estimated latency to change the service tier or rescale the compute size of a single database or elastic pool is parameterized as follows:

SERVICE TIER	BASIC SINGLE DATABASE, STANDARD (S0-S1)	BASIC ELASTIC POOL, STANDARD (S2-S12), HYPERSCALE, GENERAL PURPOSE SINGLE DATABASE OR ELASTIC POOL	PREMIUM OR BUSINESS CRITICAL SINGLE DATABASE OR ELASTIC POOL
<b>Basic single database, Standard (S0-S1)</b>	<ul style="list-style-type: none"><li>Constant time latency independent of space used</li><li>Typically, less than 5 minutes</li></ul>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>
<b>Basic elastic pool, Standard (S2-S12), Hyperscale, General Purpose single database or elastic pool</b>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>	<ul style="list-style-type: none"><li>Constant time latency independent of space used</li><li>Typically, less than 5 minutes</li></ul>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>
<b>Premium or Business Critical single database or elastic pool</b>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>	<ul style="list-style-type: none"><li>Latency proportional to database space used due to data copying</li><li>Typically, less than 1 minute per GB of space used</li></ul>

## TIP

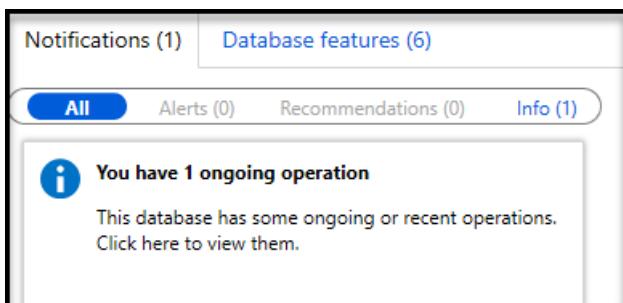
To monitor in-progress operations, see: [Manage operations using the SQL REST API](#), [Manage operations using CLI](#), [Monitor operations using T-SQL](#) and these two PowerShell commands: `Get-AzSqlDatabaseActivity` and `Stop-AzSqlDatabaseActivity`.

## Cancelling changes

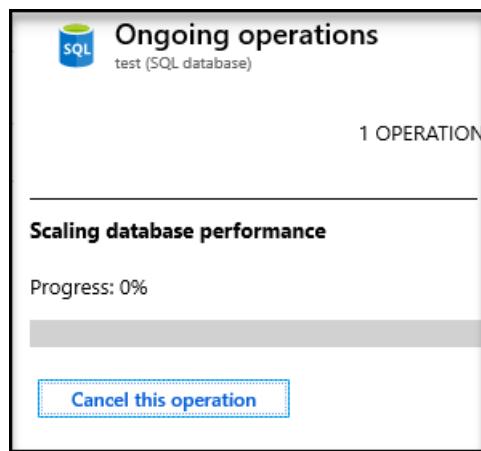
A service tier change or compute rescaling operation can be canceled.

### Azure portal

In the database overview blade, navigate to **Notifications** and click on the tile indicating there is an ongoing operation:



Next, click on the button labeled **Cancel this operation**.



## PowerShell

From a PowerShell command prompt, set the `$resourceGroupName`, `$serverName`, and `$databaseName`, and then run the following command:

```
$operationName = (az sql db op list --resource-group $resourceGroupName --server $serverName --database $databaseName --query "[?state=='InProgress'].name" --out tsv)
if (-not [string]::IsNullOrEmpty($operationName)) {
 (az sql db op cancel --resource-group $resourceGroupName --server $serverName --database $databaseName --name $operationName)
 "Operation " + $operationName + " has been canceled"
}
else {
 "No service tier change or compute rescaling operation found"
}
```

## Additional considerations

- If you are upgrading to a higher service tier or compute size, the database max size does not increase unless you explicitly specify a larger size (maxsize).
- To downgrade a database, the database used space must be smaller than the maximum allowed size of the target service tier and compute size.
- When downgrading from **Premium** to the **Standard** tier, an extra storage cost applies if both (1) the max size of the database is supported in the target compute size, and (2) the max size exceeds the included storage amount of the target compute size. For example, if a P1 database with a max size of 500 GB is downsized to S3, then an extra storage cost applies since S3 supports a max size of 1 TB and its included storage amount is only 250 GB. So, the extra storage amount is 500 GB – 250 GB = 250 GB. For pricing of extra storage, see [SQL Database pricing](#). If the actual amount of space used is less than the included storage amount, then this extra cost can be avoided by reducing the database max size to the included amount.
- When upgrading a database with [geo-replication](#) enabled, upgrade its secondary databases to the desired service tier and compute size before upgrading the primary database (general guidance for best performance). When upgrading to a different edition, it is a requirement that the secondary database is upgraded first.
- When downgrading a database with [geo-replication](#) enabled, downgrade its primary databases to the desired service tier and compute size before downgrading the secondary database (general guidance for best performance). When downgrading to a different edition, it is a requirement that the primary database is downgraded first.
- The restore service offerings are different for the various service tiers. If you are downgrading to the **Basic** tier, there is a lower backup retention period. See [Azure SQL Database Backups](#).
- The new properties for the database are not applied until the changes are complete.

## Billing

You are billed for each hour a database exists using the highest service tier + compute size that applied during that hour, regardless of usage or whether the database was active for less than an hour. For example, if you create a single database and delete it five minutes later your bill reflects a charge for one database hour.

## Change storage size

### vCore-based purchasing model

- Storage can be provisioned up to the max size limit using 1GB increments. The minimum configurable data storage is 5 GB
- Storage for a single database can be provisioned by increasing or decreasing its max size using the [Azure portal](#), [Transact-SQL](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).
- SQL Database automatically allocates 30% of additional storage for the log files and 32GB per vCore for TempDB, but not to exceed 384GB. TempDB is located on an attached SSD in all service tiers.
- The price of storage for a single database is the sum of data storage and log storage amounts multiplied by the storage unit price of the service tier. The cost of TempDB is included in the vCore price. For details on the price of extra storage, see [SQL Database pricing](#).

#### IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

### DTU-based purchasing model

- The DTU price for a single database includes a certain amount of storage at no additional cost. Extra storage beyond the included amount can be provisioned for an additional cost up to the max size limit in increments of 250 GB up to 1 TB, and then in increments of 256 GB beyond 1 TB. For included storage amounts and max size limits, see [Single database: Storage sizes and compute sizes](#).
- Extra storage for a single database can be provisioned by increasing its max size using the [Azure portal](#), [Transact-SQL](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).
- The price of extra storage for a single database is the extra storage amount multiplied by the extra storage unit price of the service tier. For details on the price of extra storage, see [SQL Database pricing](#).

#### IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

### Geo-replicated database

To change the database size of a replicated secondary database, change the size of the primary database. This change will then be replicated and implemented on the secondary database as well.

## P11 and P15 constraints when max size greater than 1 TB

More than 1 TB of storage in the Premium tier is currently available in all regions except: China East, China North, Germany Central, Germany Northeast, West Central US, US DoD regions, and US Government Central. In these regions, the storage max in the Premium tier is limited to 1 TB. The following considerations and limitations apply to P11 and P15 databases with a maximum size greater than 1 TB:

- If the max size for a P11 or P15 database was ever set to a value greater than 1 TB, then can it only be

restored or copied to a P11 or P15 database. Subsequently, the database can be rescaled to a different compute size provided the amount of space allocated at the time of the rescaling operation does not exceed max size limits of the new compute size.

- For active geo-replication scenarios:
  - Setting up a geo-replication relationship: If the primary database is P11 or P15, the secondary(ies) must also be P11 or P15; lower compute size are rejected as secondaries since they are not capable of supporting more than 1 TB.
  - Upgrading the primary database in a geo-replication relationship: Changing the maximum size to more than 1 TB on a primary database triggers the same change on the secondary database. Both upgrades must be successful for the change on the primary to take effect. Region limitations for the more than 1-TB option apply. If the secondary is in a region that does not support more than 1 TB, the primary is not upgraded.
- Using the Import/Export service for loading P11/P15 databases with more than 1 TB is not supported. Use SqlPackage.exe to [import](#) and [export](#) data.

## Next steps

For overall resource limits, see [SQL Database vCore-based resource limits - single databases](#) and [SQL Database DTU-based resource limits - elastic pools](#).

# Resource limits for elastic pools using the vCore purchasing model

1/9/2020 • 22 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database elastic pools and pooled databases using the vCore purchasing model.

For DTU purchasing model limits, see [SQL Database DTU resource limits - elastic pools](#).

## IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

You can set the service tier, compute size, and storage amount using the [Azure portal](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

## IMPORTANT

For scaling guidance and considerations, see [Scale an elastic pool](#)

## General purpose - provisioned compute - Gen4

### IMPORTANT

New Gen4 databases are no longer supported in the Australia East or Brazil South regions.

### General purpose service tier: Generation 4 compute platform (part 1)

COMPUTE SIZE	GP_GEN4_1	GP_GEN4_2	GP_GEN4_3	GP_GEN4_4	GP_GEN4_5	GP_GEN4_6
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	1	2	3	4	5	6
Memory (GB)	7	14	21	28	35	42
Max number DBs per pool	100	200	500	500	500	500
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	N/A	N/A	N/A	N/A	N/A	N/A

<b>COMPUTE SIZE</b>	<b>GP_GEN4_1</b>	<b>GP_GEN4_2</b>	<b>GP_GEN4_3</b>	<b>GP_GEN4_4</b>	<b>GP_GEN4_5</b>	<b>GP_GEN4_6</b>
Max data size (GB)	512	756	1536	1536	1536	2048
Max log size	154	227	461	461	461	614
TempDB max data size (GB)	32	64	96	128	160	192
Storage type	Premium (Remote) Storage					
IO latency (approximate )	5-7 ms (write) 5-10 ms (read)					
Max data IOPS per pool *	400	800	1200	1600	2000	2400
Max log rate per pool (MBps)	4.7	9.4	14.1	18.8	23.4	28.1
Max concurrent workers per pool (requests) **	210	420	630	840	1050	1260
Max concurrent logins per pool **	210	420	630	840	1050	1260
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1	0, 0.25, 0.5, 1, 2	0, 0.25, 0.5, 1...3	0, 0.25, 0.5, 1...4	0, 0.25, 0.5, 1...5	0, 0.25, 0.5, 1...6
Number of replicas	1	1	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A	N/A	N/A

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

General purpose service tier: Generation 4 compute platform (part 2)

<b>COMPUTE SIZE</b>	<b>GP_GEN4_7</b>	<b>GP_GEN4_8</b>	<b>GP_GEN4_9</b>	<b>GP_GEN4_10</b>	<b>GP_GEN4_16</b>	<b>GP_GEN4_24</b>
Max data IOPS per pool *	2800	3200	3600	4000	6400	9600
Max log rate per pool (MBps)	32.8	37.5	37.5	37.5	37.5	37.5
Max concurrent workers per pool (requests) *	1470	1680	1890	2100	3360	5040
Max concurrent logins pool (requests) *	1470	1680	1890	2100	3360	5040
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1...7	0, 0.25, 0.5, 1...8	0, 0.25, 0.5, 1...9	0, 0.25, 0.5, 1...10	0, 0.25, 0.5, 1...10, 16	0, 0.25, 0.5, 1...10, 16, 24
Number of replicas	1	1	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A	N/A	N/A
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size

\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

## General purpose - provisioned compute - Gen5

### General purpose service tier: Generation 5 compute platform (part 1)

<b>COMPUTE SIZE</b>	<b>GP_GEN5_2</b>	<b>GP_GEN5_4</b>	<b>GP_GEN5_6</b>	<b>GP_GEN5_8</b>	<b>GP_GEN5_10</b>	<b>GP_GEN5_12</b>	<b>GP_GEN5_14</b>
Compute generation	Gen5						
vCores	2	4	6	8	10	12	14
Memory (GB)	10.4	20.8	31.1	41.5	51.9	62.3	72.7
Max number DBs per pool	100	200	500	500	500	500	500
Columns to re support	Yes						
In-memory OLTP storage (GB)	N/A						
Max data size (GB)	512	756	1536	1536	1536	2048	2048
Max log size (GB)	154	227	461	461	461	614	614
TempDB max data size (GB)	64	128	192	256	320	384	448
Storage type	Premium (Remote) Storage						
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)						
Max data IOPS per pool *	800	1600	2400	3200	4000	4800	5600
Max log rate per pool (MBps)	9.4	18.8	28.1	37.5	37.5	37.5	37.5

<b>COMPUTE SIZE</b>	<b>GP_GEN5_2</b>	<b>GP_GEN5_4</b>	<b>GP_GEN5_6</b>	<b>GP_GEN5_8</b>	<b>GP_GEN5_10</b>	<b>GP_GEN5_12</b>	<b>GP_GEN5_14</b>
Max concurrent workers per pool (requests) **	210	420	630	840	1050	1260	1470
Max concurrent logins per pool (requests) **	210	420	630	840	1050	1260	1470
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1, 2	0, 0.25, 0.5, 1...4	0, 0.25, 0.5, 1...6	0, 0.25, 0.5, 1...8	0, 0.25, 0.5, 1...10	0, 0.25, 0.5, 1...12	0, 0.25, 0.5, 1...14
Number of replicas	1	1	1	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

#### General purpose service tier: Generation 5 compute platform (part 2)

<b>COMPUTE SIZE</b>	<b>GP_GEN5_16</b>	<b>GP_GEN5_18</b>	<b>GP_GEN5_20</b>	<b>GP_GEN5_24</b>	<b>GP_GEN5_22</b>	<b>GP_GEN5_32</b>	<b>GP_GEN5_40</b>	<b>GP_GEN5_80</b>
Compute generation	Gen5							
vCores	16	18	20	24	32	40	80	

<b>COMPUTE SIZE</b>	<b>GP_GEN5_16</b>	<b>GP_GEN5_18</b>	<b>GP_GEN5_20</b>	<b>GP_GEN5_24</b>	<b>GP_GEN5_22</b>	<b>GP_GEN5_40</b>	<b>GP_GEN5_80</b>
Memory (GB)	83	93.4	103.8	124.6	166.1	207.6	415.2
Max number DBs per pool	500	500	500	500	500	500	500
Columnstore support	Yes						
In-memory OLTP storage (GB)	N/A						
Max data size (GB)	2048	3072	3072	3072	4096	4096	4096
Max log size (GB)	614	922	922	922	1229	1229	1229
TempDB max data size (GB)	512	576	640	768	1024	1280	2560
Storage type	Premium (Remote) Storage						
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)						
Max data IOPS per pool *	6,400	7,200	8,000	9,600	12,800	16,000	32,000
Max log rate per pool (MBps)	37.5	37.5	37.5	37.5	37.5	37.5	37.5
Max concurrent workers per pool (requests) **	1680	1890	2100	2520	3360	4200	8400

<b>COMPUTE SIZE</b>	<b>GP_GEN5_6</b>	<b>GP_GEN5_8</b>	<b>GP_GEN5_20</b>	<b>GP_GEN5_24</b>	<b>GP_GEN5_22</b>	<b>GP_GEN5_40</b>	<b>GP_GEN5_80</b>
Max concurrent logins per pool (requests) **	1680	1890	2100	2520	3360	4200	8400
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000	30,000
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1...16	0, 0.25, 0.5, 1...18	0, 0.25, 0.5, 1...20	0, 0.25, 0.5, 1...20, 24	0, 0.25, 0.5, 1...20, 24, 32	0, 0.25, 0.5, 1...16, 24, 32, 40	0, 0.25, 0.5, 1...16, 24, 32, 40, 80
Number of replicas	1	1	1	1	1	1	1
Multi-AZ	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Read Scale-out	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

## General purpose - provisioned compute - Fsv2-series

### Fsv2-series compute generation (preview)

<b>COMPUTE SIZE</b>	<b>GP_FSV2_72</b>
Compute generation	Fsv2-series
vCores	72
Memory (GB)	136.2
Max number DBs per pool	500

COMPUTE SIZE	GP_FSV2_72
Columnstore support	Yes
In-memory OLTP storage (GB)	N/A
Max data size (GB)	4096
Max log size (GB)	1024
TempDB max data size (GB)	333
Storage type	Premium (Remote) Storage
IO latency (approximate)	5-7 ms (write) 5-10 ms (read)
Max data IOPS per pool *	16,000
Max log rate per pool (MBps)	37.5
Max concurrent workers per pool (requests) **	3780
Max concurrent logins per pool (requests) **	3780
Max concurrent sessions	30,000
Min/max elastic pool vCore choices per database	0-72
Number of replicas	1
Multi-AZ	N/A
Read Scale-out	N/A
Included backup storage	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent.  
For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#).  
For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

## Business critical - provisioned compute - Gen4

### IMPORTANT

New Gen4 databases are no longer supported in the Australia East or Brazil South regions.

### Business critical service tier: Generation 4 compute platform (part 1)

<b>COMPUTE SIZE</b>	<b>BC_GEN4_2</b>	<b>BC_GEN4_3</b>	<b>BC_GEN4_4</b>	<b>BC_GEN4_5</b>	<b>BC_GEN4_6</b>
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	2	3	4	5	6
Memory (GB)	14	21	28	35	42
Max number DBs per pool	100	100	100	100	100
Columnstore support	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	2	3	4	5	6
Storage type	Local SSD				
Max data size (GB)	1024	1024	1024	1024	1024
Max log size (GB)	307	307	307	307	307
TempDB max data size (GB)	64	96	128	160	192
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)				
Max data IOPS per pool *	9,000	13,500	18,000	22,500	27,000
Max log rate per pool (MBps)	20	30	40	50	60
Max concurrent workers per pool (requests) **	420	630	840	1050	1260
Max concurrent logins per pool (requests) **	420	630	840	1050	1260
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000

Compute size	BC_GEN4_2	BC_GEN4_3	BC_GEN4_4	BC_GEN4_5	BC_GEN4_6
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1, 2	0, 0.25, 0.5, 1...3	0, 0.25, 0.5, 1...4	0, 0.25, 0.5, 1...5	0, 0.25, 0.5, 1...6
Number of replicas	4	4	4	4	4
Multi-AZ	Yes	Yes	Yes	Yes	Yes
Read Scale-out	Yes	Yes	Yes	Yes	Yes
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

#### Business critical service tier: Generation 4 compute platform (part 2)

Compute size	BC_GEN4_7	BC_GEN4_8	BC_GEN4_9	BC_GEN4_10	BC_GEN4_16	BC_GEN4_24
Compute generation	Gen4	Gen4	Gen4	Gen4	Gen4	Gen4
vCores	7	8	9	10	16	24
Memory (GB)	49	56	63	70	112	159.5
Max number DBs per pool	100	100	100	100	100	100
Columnstore support	N/A	N/A	N/A	N/A	N/A	N/A
In-memory OLTP storage (GB)	7	8	9.5	11	20	36
Storage type	Local SSD	Local SSD	Local SSD	Local SSD	Local SSD	Local SSD
Max data size (GB)	1024	1024	1024	1024	1024	1024
Max log size (GB)	307	307	307	307	307	307

Compute Size	BC_GEN4_7	BC_GEN4_8	BC_GEN4_9	BC_GEN4_10	BC_GEN4_16	BC_GEN4_24
TempDB max data size (GB)	224	256	288	320	512	768
IO latency (approximate )	1-2 ms (write) 1-2 ms (read)					
Max data IOPS per pool *	31,500	36,000	40,500	45,000	72,000	96,000
Max log rate per pool (MBps)	70	80	80	80	80	80
Max concurrent workers per pool (requests) **	1470	1680	1890	2100	3360	5040
Max concurrent logins per pool (requests) **	1470	1680	1890	2100	3360	5040
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1...7	0, 0.25, 0.5, 1...8	0, 0.25, 0.5, 1...9	0, 0.25, 0.5, 1...10	0, 0.25, 0.5, 1...10, 16	0, 0.25, 0.5, 1...10, 16, 24
Number of replicas	4	4	4	4	4	4
Multi-AZ	Yes	Yes	Yes	Yes	Yes	Yes
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes
Included backup storage	1X DB size					

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore

settings per database that are less than 1 vCore or less, the number of max concurrent workers is similarly rescaled.

## Business critical - provisioned compute - Gen5

### Business critical service tier: Generation 5 compute platform (part 1)

COMPUTE SIZE	BC_GEN5_4	BC_GEN5_6	BC_GEN5_8	BC_GEN5_10	BC_GEN5_12	BC_GEN5_14
Compute generation	Gen5	Gen5	Gen5	Gen5	Gen5	Gen5
vCores	4	6	8	10	12	14
Memory (GB)	20.8	31.1	41.5	51.9	62.3	72.7
Max number DBs per pool	100	100	100	100	100	100
Columnstore support	Yes	Yes	Yes	Yes	Yes	Yes
In-memory OLTP storage (GB)	3.14	4.71	6.28	8.65	11.02	13.39
Max data size (GB)	1024	1536	1536	1536	3072	3072
Max log size (GB)	307	307	461	461	922	922
TempDB max data size (GB)	128	192	256	320	384	448
Storage type	Local SSD					
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)					
Max data IOPS per pool *	18,000	27,000	36,000	45,000	54,000	63,000
Max log rate per pool (MBps)	60	90	120	120	120	120
Max concurrent workers per pool (requests) **	420	630	840	1050	1260	1470

Compute size	BC_GEN5_4	BC_GEN5_6	BC_GEN5_8	BC_GEN5_10	BC_GEN5_12	BC_GEN5_14
Max concurrent logins per pool (requests) **	420	630	840	1050	1260	1470
Max concurrent sessions	30,000	30,000	30,000	30,000	30,000	30,000
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1...4	0, 0.25, 0.5, 1...6	0, 0.25, 0.5, 1...8	0, 0.25, 0.5, 1...10	0, 0.25, 0.5, 1...12	0, 0.25, 0.5, 1...14
Number of replicas	4	4	4	4	4	4
Multi-AZ	Yes	Yes	Yes	Yes	Yes	Yes
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

### Business critical service tier: Generation 5 compute platform (part 2)

Compute size	BC_GEN5_16	BC_GEN5_18	BC_GEN5_20	BC_GEN5_24	BC_GEN5_32	BC_GEN5_40	BC_GEN5_80
Compute generation	Gen5						
vCores	16	18	20	24	32	40	80
Memory (GB)	83	93.4	103.8	124.6	166.1	207.6	415.2
Max number DBs per pool	100	100	100	100	100	100	100



<b>COMPUTE SIZE</b>	<b>BC_GEN5_16</b>	<b>BC_GEN5_18</b>	<b>BC_GEN5_20</b>	<b>BC_GEN5_24</b>	<b>BC_GEN5_32</b>	<b>BC_GEN5_40</b>	<b>BC_GEN5_80</b>
Min/max elastic pool vCore choices per database	0, 0.25, 0.5, 1...16	0, 0.25, 0.5, 1...18	0, 0.25, 0.5, 1...20	0, 0.25, 0.5, 1...20, 24	0, 0.25, 0.5, 1...20, 24, 32	0, 0.25, 0.5, 1...20, 24, 32, 40	0, 0.25, 0.5, 1...20, 24, 32, 40, 80
Number of replicas	4	4	4	4	4	4	4
Multi-AZ	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Read Scale-out	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Included backup storage	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

## Business critical - provisioned compute - M-series

### M-series compute generation (preview)

<b>COMPUTE SIZE</b>	<b>BC_M_128</b>
Compute generation	M-series
vCores	128
Memory (GB)	3767.1
Max number DBs per pool	100
Columnstore support	Yes
In-memory OLTP storage (GB)	1768
Max data size (GB)	4096
Max log size (GB)	2048
TempDB max data size (GB)	4096

COMPUTE SIZE	BC_M_128
Storage type	Local SSD
IO latency (approximate)	1-2 ms (write) 1-2 ms (read)
Max data IOPS per pool *	200,000
Max log rate per pool (MBps)	333
Max concurrent workers per pool (requests) *	13,440
Max concurrent logins per pool (requests) *	13,440
Max concurrent sessions	30,000
Min/max elastic pool vCore choices per database	0-128
Number of replicas	4
Multi-AZ	Yes
Read Scale-out	Yes
Included backup storage	1X DB size

\* The maximum value for IO sizes ranging between 8 KB and 64 KB. Actual IOPS are workload-dependent. For details, see [Data IO Governance](#).

\*\* For the max concurrent workers (requests) for any individual database, see [Single database resource limits](#). For example, if the elastic pool is using Gen5 and the max vCore per database is set at 2, then the max concurrent workers value is 200. If max vCore per database is set to 0.5, then the max concurrent workers value is 50 since on Gen5 there are a max of 100 concurrent workers per vCore. For other max vCore settings per database that are less 1 vCore or less, the number of max concurrent workers is similarly rescaled.

If all vCores of an elastic pool are busy, then each database in the pool receives an equal amount of compute resources to process queries. The SQL Database service provides resource sharing fairness between databases by ensuring equal slices of compute time. Elastic pool resource sharing fairness is in addition to any amount of resource otherwise guaranteed to each database when the vCore min per database is set to a non-zero value.

## Database properties for pooled databases

The following table describes the properties for pooled databases.

### NOTE

The resource limits of individual databases in elastic pools are generally the same as for single databases outside of pools that has the same compute size. For example, the max concurrent workers for an GP\_Gen4\_1 database is 200 workers. So, the max concurrent workers for a database in a GP\_Gen4\_1 pool is also 200 workers. Note, the total number of concurrent workers in GP\_Gen4\_1 pool is 210.

PROPERTY	DESCRIPTION
Max vCores per database	The maximum number of vCores that any database in the pool may use, if available based on utilization by other databases in the pool. Max vCores per database is not a resource guarantee for a database. This setting is a global setting that applies to all databases in the pool. Set max vCores per database high enough to handle peaks in database utilization. Some degree of over-committing is expected since the pool generally assumes hot and cold usage patterns for databases where all databases are not simultaneously peaking.
Min vCores per database	The minimum number of vCores that any database in the pool is guaranteed. This setting is a global setting that applies to all databases in the pool. The min vCores per database may be set to 0, and is also the default value. This property is set to anywhere between 0 and the average vCores utilization per database. The product of the number of databases in the pool and the min vCores per database cannot exceed the vCores per pool.
Max storage per database	The maximum database size set by the user for a database in a pool. Pooled databases share allocated pool storage, so the size a database can reach is limited to the smaller of remaining pool storage and database size. Max database size refers to the maximum size of the data files and does not include the space used by log files.

## Next steps

- For vCore resource limits for a single database, see [resource limits for single databases using the vCore purchasing model](#)
- For DTU resource limits for a single database, see [resource limits for single databases using the DTU purchasing model](#)
- For DTU resource limits for elastic pools, see [resource limits for elastic pools using the DTU purchasing model](#)
- For resource limits for managed instances, see [managed instance resource limits](#).
- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).
- For information about resource limits on a database server, see [overview of resource limits on a SQL Database server](#) for information about limits at the server and subscription levels.

# Resources limits for elastic pools using the DTU purchasing model

1/17/2020 • 8 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database elastic pools and pooled databases using the DTU purchasing model.

For DTU purchasing model resource limits for single databases, see [DTU resource limits - single databases](#). For vCore resource limits, see [vCore resource limits - single databases](#) and [vCore resource limits - elastic pools](#).

## Elastic pool: Storage sizes and compute sizes

For SQL Database elastic pools, the following tables show the resources available at each service tier and compute size. You can set the service tier, compute size, and storage amount using the [Azure portal](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

**IMPORTANT**

For scaling guidance and considerations, see [Scale an elastic pool](#)

## **NOTE**

The resource limits of individual databases in elastic pools are generally the same as for single databases outside of pools based on DTUs and the service tier. For example, the max concurrent workers for an S2 database is 120 workers. So, the max concurrent workers for a database in a Standard pool is also 120 workers if the max DTU per database in the pool is 50 DTUs (which is equivalent to S2).

## Basic elastic pool limits

<b>EDTUS PER POOL</b>	<b>50</b>	<b>100</b>	<b>200</b>	<b>300</b>	<b>400</b>	<b>800</b>	<b>1200</b>	<b>1600</b>
Max number DBs per pool	100	200	500	500	500	500	500	500
Max concurrent workers (requests) per pool	100	200	400	600	800	1600	2400	3200
Max concurrent sessions per pool	30000	30000	30000	30000	30000	30000	30000	30000
Min eDTUs choices per database	0, 5	0, 5	0, 5	0, 5	0, 5	0, 5	0, 5	0, 5
Max eDTUs choices per database	5	5	5	5	5	5	5	5
Max storage per database (GB)	2	2	2	2	2	2	2	2

### Standard elastic pool limits

<b>EDTUS PER POOL</b>	<b>50</b>	<b>100</b>	<b>200</b>	<b>300</b>	<b>400</b>	<b>800</b>
Included storage per pool (GB)	50	100	200	300	400	800
Max storage choices per pool (GB)	50, 250, 500	100, 250, 500, 750	200, 250, 500, 750, 1024	300, 500, 750, 1024, 1280	400, 500, 750, 1024, 1280, 1536	800, 1024, 1280, 1536, 1792, 2048
Max In-Memory OLTP storage per pool (GB)	N/A	N/A	N/A	N/A	N/A	N/A

<b>EDTUS PER POOL</b>	<b>50</b>	<b>100</b>	<b>200</b>	<b>300</b>	<b>400</b>	<b>800</b>
Max number DBs per pool	100	200	500	500	500	500
Max concurrent workers (requests) per pool	100	200	400	600	800	1600
Max concurrent sessions per pool	30000	30000	30000	30000	30000	30000
Min eDTUs choices per database	0, 10, 20, 50	0, 10, 20, 50, 100	0, 10, 20, 50, 100, 200	0, 10, 20, 50, 100, 200, 300	0, 10, 20, 50, 100, 200, 300, 400	0, 10, 20, 50, 100, 200, 300, 400, 800
Max eDTUs choices per database	10, 20, 50	10, 20, 50, 100	10, 20, 50, 100, 200	10, 20, 50, 100, 200, 300	10, 20, 50, 100, 200, 300, 400	10, 20, 50, 100, 200, 300, 400, 800
Max storage per database (GB)	500	750	1024	1024	1024	1024

#### Standard elastic pool limits (continued)

<b>EDTUS PER POOL</b>	<b>1200</b>	<b>1600</b>	<b>2000</b>	<b>2500</b>	<b>3000</b>
Included storage per pool (GB)	1200	1600	2000	2500	3000
Max storage choices per pool (GB)	1200, 1280, 1536, 1792, 2048, 2304, 2560	1600, 1792, 2048, 2304, 2560, 2816, 3072	2000, 2048, 2304, 2560, 2816, 3072, 3328, 3584	2500, 2560, 2816, 3072, 3328, 3584, 3840, 4096	3000, 3072, 3328, 3584, 3840, 4096
Max In-Memory OLTP storage per pool (GB)	N/A	N/A	N/A	N/A	N/A
Max number DBs per pool	500	500	500	500	500
Max concurrent workers (requests) per pool	2400	3200	4000	5000	6000
Max concurrent sessions per pool	30000	30000	30000	30000	30000

<b>EDTUS PER POOL</b>	<b>1200</b>	<b>1600</b>	<b>2000</b>	<b>2500</b>	<b>3000</b>
Min eDTUs choices per database	0, 10, 20, 50, 100, 200, 300, 400, 800, 1200	0, 10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600	0, 10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600, 2000	0, 10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600, 2000, 2500	0, 10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600, 2000, 2500, 3000
Max eDTUs choices per database	10, 20, 50, 100, 200, 300, 400, 800, 1200	10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600	10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600, 2000	10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600, 2000, 2500	10, 20, 50, 100, 200, 300, 400, 800, 1200, 1600, 2000, 2500, 3000
Max storage choices per database (GB)	1024	1024	1024	1024	1024

### Premium elastic pool limits

<b>EDTUS PER POOL</b>	<b>125</b>	<b>250</b>	<b>500</b>	<b>1000</b>	<b>1500</b>
Included storage per pool (GB)	250	500	750	1024	1536
Max storage choices per pool (GB)	250, 500, 750, 1024	500, 750, 1024	750, 1024	1024	1536
Max In-Memory OLTP storage per pool (GB)	1	2	4	10	12
Max number DBs per pool	50	100	100	100	100
Max concurrent workers per pool (requests)	200	400	800	1600	2400
Max concurrent sessions per pool	30000	30000	30000	30000	30000
Min eDTUs per database	0, 25, 50, 75, 125	0, 25, 50, 75, 125, 250	0, 25, 50, 75, 125, 250, 500	0, 25, 50, 75, 125, 250, 500, 1000	0, 25, 50, 75, 125, 250, 500, 1000
Max eDTUs per database	25, 50, 75, 125	25, 50, 75, 125, 250	25, 50, 75, 125, 250, 500	25, 50, 75, 125, 250, 500, 1000	25, 50, 75, 125, 250, 500, 1000
Max storage per database (GB)	1024	1024	1024	1024	1024

### Premium elastic pool limits (continued)

<b>EDTUS PER POOL</b>	<b>2000</b>	<b>2500</b>	<b>3000</b>	<b>3500</b>	<b>4000</b>
Included storage per pool (GB)	2048	2560	3072	3548	4096
Max storage choices per pool (GB)	2048	2560	3072	3548	4096
Max In-Memory OLTP storage per pool (GB)	16	20	24	28	32
Max number DBs per pool	100	100	100	100	100
Max concurrent workers (requests) per pool	3200	4000	4800	5600	6400
Max concurrent sessions per pool	30000	30000	30000	30000	30000
Min eDTUs choices per database	0, 25, 50, 75, 125, 250, 500, 1000, 1750	0, 25, 50, 75, 125, 250, 500, 1000, 1750	0, 25, 50, 75, 125, 250, 500, 1000, 1750	0, 25, 50, 75, 125, 250, 500, 1000, 1750	0, 25, 50, 75, 125, 250, 500, 1000, 1750, 4000
Max eDTUs choices per database	25, 50, 75, 125, 250, 500, 1000, 1750	25, 50, 75, 125, 250, 500, 1000, 1750	25, 50, 75, 125, 250, 500, 1000, 1750	25, 50, 75, 125, 250, 500, 1000, 1750	25, 50, 75, 125, 250, 500, 1000, 1750, 4000
Max storage per database (GB)	1024	1024	1024	1024	1024

### IMPORTANT

More than 1 TB of storage in the Premium tier is currently available in all regions except: China East, China North, Germany Central, Germany Northeast, West Central US, US DoD regions, and US Government Central. In these regions, the storage max in the Premium tier is limited to 1 TB. For more information, see [P11-P15 current limitations](#).

If all DTUs of an elastic pool are used, then each database in the pool receives an equal amount of resources to process queries. The SQL Database service provides resource sharing fairness between databases by ensuring equal slices of compute time. Elastic pool resource sharing fairness is in addition to any amount of resource otherwise guaranteed to each database when the DTU min per database is set to a non-zero value.

### NOTE

For `tempdb` limits, see [tempdb limits](#).

## Database properties for pooled databases

The following table describes the properties for pooled databases.

PROPERTY	DESCRIPTION
Max eDTUs per database	The maximum number of eDTUs that any database in the pool may use, if available based on utilization by other databases in the pool. Max eDTU per database is not a resource guarantee for a database. This setting is a global setting that applies to all databases in the pool. Set max eDTUs per database high enough to handle peaks in database utilization. Some degree of overcommitting is expected since the pool generally assumes hot and cold usage patterns for databases where all databases are not simultaneously peaking. For example, suppose the peak utilization per database is 20 eDTUs and only 20% of the 100 databases in the pool are peak at the same time. If the eDTU max per database is set to 20 eDTUs, then it is reasonable to overcommit the pool by 5 times, and set the eDTUs per pool to 400.
Min eDTUs per database	The minimum number of eDTUs that any database in the pool is guaranteed. This setting is a global setting that applies to all databases in the pool. The min eDTU per database may be set to 0, and is also the default value. This property is set to anywhere between 0 and the average eDTU utilization per database. The product of the number of databases in the pool and the min eDTUs per database cannot exceed the eDTUs per pool. For example, if a pool has 20 databases and the eDTU min per database set to 10 eDTUs, then the eDTUs per pool must be at least as large as 200 eDTUs.
Max storage per database	The maximum database size set by the user for a database in a pool. However, pooled databases share allocated pool storage. Even if the total max storage <i>per database</i> is set to be greater than the total available storage <i>space of the pool</i> , the total space actually used by all of the databases will not be able to exceed the available pool limit. Max database size refers to the maximum size of the data files and does not include the space used by log files.

## Next steps

- For vCore resource limits for a single database, see [resource limits for single databases using the vCore purchasing model](#)
- For DTU resource limits for a single database, see [resource limits for single databases using the DTU purchasing model](#)
- For vCore resource limits for elastic pools, see [resource limits for elastic pools using the vCore purchasing model](#)
- For resource limits for managed instances, see [managed instance resource limits](#).
- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).
- For information about resource limits on a database server, see [overview of resource limits on a SQL Database server](#) for information about limits at the server and subscription levels.

# Scale elastic pool resources in Azure SQL Database

2/18/2020 • 6 minutes to read • [Edit Online](#)

This article describes how to scale the compute and storage resources available for elastic pools and pooled databases in Azure SQL Database.

## Change compute resources (vCores or DTUs)

After initially picking the number of vCores or eDTUs, you can scale an elastic pool up or down dynamically based on actual experience using the [Azure portal](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

### Impact of changing service tier or rescaling compute size

Changing the service tier or compute size of an elastic pool follows a similar pattern as for single databases and mainly involves the service performing the following steps:

1. Create new compute instance for the elastic pool

A new compute instance for the elastic pool is created with the requested service tier and compute size.

For some combinations of service tier and compute size changes, a replica of each database must be created in the new compute instance which involves copying data and can strongly influence the overall latency. Regardless, the databases remain online during this step, and connections continue to be directed to the databases in the original compute instance.

2. Switch routing of connections to new compute instance

Existing connections to the databases in the original compute instance are dropped. Any new connections are established to the databases in the new compute instance. For some combinations of service tier and compute size changes, database files are detached and reattached during the switch. Regardless, the switch can result in a brief service interruption when databases are unavailable generally for less than 30 seconds and often for only a few seconds. If there are long running transactions running when connections are dropped, the duration of this step may take longer in order to recover aborted transactions. [Accelerated Database Recovery](#) can reduce the impact from aborting long running transactions.

#### IMPORTANT

No data is lost during any step in the workflow.

### Latency of changing service tier or rescaling compute size

The estimated latency to change the service tier or rescale the compute size of a single database or elastic pool is parameterized as follows:

SERVICE TIER	BASIC SINGLE DATABASE, STANDARD (S0-S1)	BASIC ELASTIC POOL, STANDARD (S2-S12), HYPERSCALE, GENERAL PURPOSE SINGLE DATABASE OR ELASTIC POOL	PREMIUM OR BUSINESS CRITICAL SINGLE DATABASE OR ELASTIC POOL
--------------	--------------------------------------------	----------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------

Service tier	Basic single database, Standard (S0-S1)	Basic elastic pool, Standard (S2-S12), Hyperscale, General purpose single database or elastic pool	Premium or business critical single database or elastic pool
<b>Basic single database, Standard (S0-S1)</b>	<ul style="list-style-type: none"> <li>Constant time latency independent of space used</li> <li>Typically, less than 5 minutes</li> </ul>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>
<b>Basic elastic pool, Standard (S2-S12), Hyperscale, General Purpose single database or elastic pool</b>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>	<ul style="list-style-type: none"> <li>Constant time latency independent of space used</li> <li>Typically, less than 5 minutes</li> </ul>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>
<b>Premium or Business Critical single database or elastic pool</b>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>	<ul style="list-style-type: none"> <li>Latency proportional to database space used due to data copying</li> <li>Typically, less than 1 minute per GB of space used</li> </ul>

#### NOTE

- In the case of changing the service tier or rescaling compute for an elastic pool, the summation of space used across all databases in the pool should be used to calculate the estimate.
- In the case of moving a database to/from an elastic pool, only the space used by the database impacts the latency, not the space used by the elastic pool.

#### TIP

To monitor in-progress operations, see: [Manage operations using the SQL REST API](#), [Manage operations using CLI](#), [Monitor operations using T-SQL](#) and these two PowerShell commands: `Get-AzSqlDatabaseActivity` and `Stop-AzSqlDatabaseActivity`.

#### Additional considerations when changing service tier or rescaling compute size

- When downsizing vCores or eDTUs for an elastic pool, the pool used space must be smaller than the maximum allowed size of the target service tier and pool eDTUs.
- When rescaling eDTUs for an elastic pool, an extra storage cost applies if (1) the storage max size of the pool is supported by the target pool, and (2) the storage max size exceeds the included storage amount of the target pool. For example, if a 100 eDTU Standard pool with a max size of 100 GB is downsized to a 50 eDTU Standard pool, then an extra storage cost applies since target pool supports a max size of 100 GB and its included storage amount is only 50 GB. So, the extra storage amount is  $100\text{ GB} - 50\text{ GB} = 50\text{ GB}$ . For pricing of extra storage, see [SQL Database pricing](#). If the actual amount of space used is less than the included storage amount, then this extra cost can be avoided by reducing the database max size to the included amount.

#### Billing during rescaling

You are billed for each hour a database exists using the highest service tier + compute size that applied during that hour, regardless of usage or whether the database was active for less than an hour. For example, if you

create a single database and delete it five minutes later your bill reflects a charge for one database hour.

## Change elastic pool storage size

### IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

### vCore-based purchasing model

- Storage can be provisioned up to the max size limit:
  - For storage in the standard or general purpose service tiers, increase or decrease size in 10-GB increments
  - For storage in the premium or business critical service tiers, increase or decrease size in 250-GB increments
- Storage for an elastic pool can be provisioned by increasing or decreasing its max size.
- The price of storage for an elastic pool is the storage amount multiplied by the storage unit price of the service tier. For details on the price of extra storage, see [SQL Database pricing](#).

### IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

### DTU-based purchasing model

- The eDTU price for an elastic pool includes a certain amount of storage at no additional cost. Extra storage beyond the included amount can be provisioned for an additional cost up to the max size limit in increments of 250 GB up to 1 TB, and then in increments of 256 GB beyond 1 TB. For included storage amounts and max size limits, see [Elastic pool: storage sizes and compute sizes](#).
- Extra storage for an elastic pool can be provisioned by increasing its max size using the [Azure portal](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).
- The price of extra storage for an elastic pool is the extra storage amount multiplied by the extra storage unit price of the service tier. For details on the price of extra storage, see [SQL Database pricing](#).

### IMPORTANT

Under some circumstances, you may need to shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#).

## Next steps

For overall resource limits, see [SQL Database vCore-based resource limits - elastic pools](#) and [SQL Database DTU-based resource limits - elastic pools](#).

# Manage elastic pools in Azure SQL Database

11/7/2019 • 5 minutes to read • [Edit Online](#)

With an elastic pool, you determine the amount of resources that the elastic pool requires to handle the workload of its databases, and the amount of resources for each pooled database.

## Azure portal: Manage elastic pools and pooled databases

All pool settings can be found in one place: the **Configure pool** blade. To get here, find an elastic pool in the portal and click **Configure pool** either from the top of the blade or from the resource menu on the left.

From here you can make any combination of the following changes and save them all in one batch:

1. Change the service tier of the pool
2. Scale the performance (DTU or vCores) and storage up or down
3. Add or remove databases to/from the pool
4. Set a min (guaranteed) and max performance limit for the databases in the pools
5. Review the cost summary to view any changes to your bill as a result of your new selections

The screenshot shows the 'Pool1 - Configure pool' blade in the Azure portal. On the left, there's a sidebar with various navigation links like Overview, Activity log, Tags, and Configure pool (which is selected). The main content area has three tabs at the top: Pool settings (selected), Databases, and Per database settings. Under Pool settings, there are buttons for '+ Add database', 'Remove from pool', and 'Revert selected'. Below these are sections for 'Databases to be removed from pool', 'Ready to be added to this pool', and 'Currently in this pool'. A 'Select all' checkbox and a 'Columns' button are also present. A search bar at the bottom says 'Search to filter databases...'. To the right, there are two main sections: 'General Purpose' and 'Business Critical'. Each section has a description, vCore requirements, and a starting price. At the bottom right, there's a 'Cost Summary' table:

Gen4 - General Purpose	210.27
Cost per vCore (in USD)	x 1
Cost per GB (in USD)	0.14
Max storage selected (in GB)	x 41.6
EST. COST PER MONTH	216.02 USD

## PowerShell: Manage elastic pools and pooled databases

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

To create and manage SQL Database elastic pools and pooled databases with Azure PowerShell, use the following PowerShell cmdlets. If you need to install or upgrade PowerShell, see [Install Azure PowerShell module](#). To create and manage the SQL Database servers for an elastic pool, see [Create and manage SQL Database servers](#). To create and manage firewall rules, see [Create and manage firewall rules using PowerShell](#).

## TIP

For PowerShell example scripts, see [Create elastic pools and move databases between pools and out of a pool using PowerShell](#) and [Use PowerShell to monitor and scale a SQL elastic pool in Azure SQL Database](#).

CMDLET	DESCRIPTION
<a href="#">New-AzSqlElasticPool</a>	Creates an elastic pool.
<a href="#">Get-AzSqlElasticPool</a>	Gets elastic pools and their property values.
<a href="#">Set-AzSqlElasticPool</a>	Modifies properties of an elastic pool For example, use the <b>StorageMB</b> property to modify the max storage of an elastic pool.
<a href="#">Remove-AzSqlElasticPool</a>	Deletes an elastic pool.
<a href="#">Get-AzSqlElasticPoolActivity</a>	Gets the status of operations on an elastic pool
<a href="#">New-AzSqlDatabase</a>	Creates a new database in an existing pool or as a single database.
<a href="#">Get-AzSqlDatabase</a>	Gets one or more databases.
<a href="#">Set-AzSqlDatabase</a>	Sets properties for a database, or moves an existing database into, out of, or between elastic pools.
<a href="#">Remove-AzSqlDatabase</a>	Removes a database.

## TIP

Creation of many databases in an elastic pool can take time when done using the portal or PowerShell cmdlets that create only a single database at a time. To automate creation into an elastic pool, see [CreateOrUpdateElasticPoolAndPopulate](#).

## Azure CLI: Manage elastic pools and pooled databases

To create and manage SQL Database elastic pools with the [Azure CLI](#), use the following [Azure CLI SQL Database](#) commands. Use the [Cloud Shell](#) to run the CLI in your browser, or [install](#) it on macOS, Linux, or Windows.

**TIP**

For Azure CLI example scripts, see [Use CLI to move an Azure SQL database in a SQL elastic pool](#) and [Use Azure CLI to scale a SQL elastic pool in Azure SQL Database](#).

CMDLET	DESCRIPTION
<a href="#">az sql elastic-pool create</a>	Creates an elastic pool.
<a href="#">az sql elastic-pool list</a>	Returns a list of elastic pools in a server.
<a href="#">az sql elastic-pool list-dbs</a>	Returns a list of databases in an elastic pool.
<a href="#">az sql elastic-pool list-editions</a>	Also includes available pool DTU settings, storage limits, and per database settings. In order to reduce verbosity, additional storage limits and per database settings are hidden by default.
<a href="#">az sql elastic-pool update</a>	Updates an elastic pool.
<a href="#">az sql elastic-pool delete</a>	Deletes the elastic pool.

## Transact-SQL: Manage pooled databases

To create and move databases within existing elastic pools or to return information about an SQL Database elastic pool with Transact-SQL, use the following T-SQL commands. You can issue these commands using the Azure portal, [SQL Server Management Studio](#), [Visual Studio Code](#), or any other program that can connect to an Azure SQL Database server and pass Transact-SQL commands. To create and manage firewall rules using T-SQL, see [Manage firewall rules using Transact-SQL](#).

**IMPORTANT**

You cannot create, update, or delete an Azure SQL Database elastic pool using Transact-SQL. You can add or remove databases from an elastic pool, and you can use DMVs to return information about existing elastic pools.

COMMAND	DESCRIPTION
<a href="#">CREATE DATABASE (Azure SQL Database)</a>	Creates a new database in an existing pool or as a single database. You must be connected to the master database to create a new database.
<a href="#">ALTER DATABASE (Azure SQL Database)</a>	Move a database into, out of, or between elastic pools.
<a href="#">DROP DATABASE (Transact-SQL)</a>	Deletes a database.
<a href="#">sys.elastic_pool_resource_stats (Azure SQL Database)</a>	Returns resource usage statistics for all the elastic pools in a SQL Database server. For each elastic pool, there is one row for each 15 second reporting window (four rows per minute). This includes CPU, IO, Log, storage consumption and concurrent request/session utilization by all databases in the pool.

COMMAND	DESCRIPTION
<a href="#">sys.database_service_objectives (Azure SQL Database)</a>	Returns the edition (service tier), service objective (pricing tier), and elastic pool name, if any, for an Azure SQL database or an Azure SQL Data Warehouse. If logged on to the master database in an Azure SQL Database server, returns information on all databases. For Azure SQL Data Warehouse, you must be connected to the master database.

## REST API: Manage elastic pools and pooled databases

To create and manage SQL Database elastic pools and pooled databases, use these REST API requests.

COMMAND	DESCRIPTION
<a href="#">Elastic pools - Create or update</a>	Creates a new elastic pool or updates an existing elastic pool.
<a href="#">Elastic pools - Delete</a>	Deletes the elastic pool.
<a href="#">Elastic pools - Get</a>	Gets an elastic pool.
<a href="#">Elastic pools - List by server</a>	Returns a list of elastic pools in a server.
<a href="#">Elastic pools - Update</a>	Updates an existing elastic pool.
<a href="#">Elastic pool activities</a>	Returns elastic pool activities.
<a href="#">Elastic pool database activities</a>	Returns activity on databases inside of an elastic pool.
<a href="#">Databases - Create or update</a>	Creates a new database or updates an existing database.
<a href="#">Databases - Get</a>	Gets a database.
<a href="#">Databases - List by elastic pool</a>	Returns a list of databases in an elastic pool.
<a href="#">Databases - List by server</a>	Returns a list of databases in a server.
<a href="#">Databases - Update</a>	Updates an existing database.

## Next steps

- To learn more about design patterns for SaaS applications using elastic pools, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).
- For a SaaS tutorial using elastic pools, see [Introduction to the Wingtip SaaS application](#).

# SQL Database resource limits and resource governance

2/24/2020 • 9 minutes to read • [Edit Online](#)

This article provides an overview of the SQL Database resource limits for a SQL Database server that manages single databases and elastic pools. It provides information on what happens when those resource limits are hit or exceeded, and describes the resource governance mechanisms used to enforce these limits.

## NOTE

For Managed Instance limits, see [SQL Database resource limits for managed instances](#).

## Maximum resource limits

RESOURCE	LIMIT
Databases per server	5000
Default number of servers per subscription in any region	20
Max number of servers per subscription in any region	200
DTU / eDTU quota per server	54,000
vCore quota per server/instance	540
Max pools per server	Limited by number of DTUs or vCores. For example, if each pool is 1000 DTUs, then a server can support 54 pools.

## IMPORTANT

As the number of databases approaches the limit per SQL Database server, the following can occur:

- Increasing latency in running queries against the master database. This includes views of resource utilization statistics such as `sys.resource_stats`.
- Increasing latency in management operations and rendering portal viewpoints that involve enumerating databases in the server.

## NOTE

To obtain more DTU/eDTU quota, vCore quota, or more servers than the default amount, submit a new support request in the Azure portal. For more information, see [Request quota increases for Azure SQL Database](#).

## Storage size

For single databases resource storage sizes, refer to either [DTU-based resource limits](#) or [vCore-based resource limits](#) for the storage size limits per pricing tier.

# What happens when database resource limits are reached

## Compute (DTUs and eDTUs / vCores)

When database compute utilization (measured by DTUs and eDTUs, or vCores) becomes high, query latency increases, and queries can even time out. Under these conditions, queries may be queued by the service and are provided resources for execution as resources become free. When encountering high compute utilization, mitigation options include:

- Increasing the compute size of the database or elastic pool to provide the database with more compute resources. See [Scale single database resources](#) and [Scale elastic pool resources](#).
- Optimizing queries to reduce the resource utilization of each query. For more information, see [Query Tuning/Hinting](#).

## Storage

When database space used reaches the max size limit, database inserts and updates that increase the data size fail and clients receive an [error message](#). SELECT and DELETE statements continue to succeed.

When encountering high space utilization, mitigation options include:

- Increasing the max size of the database or elastic pool, or adding more storage. See [Scale single database resources](#) and [Scale elastic pool resources](#).
- If the database is in an elastic pool, then alternatively the database can be moved outside of the pool so that its storage space is not shared with other databases.
- Shrink a database to reclaim unused space. For more information, see [Manage file space in Azure SQL Database](#)

## Sessions and workers (requests)

The maximum number of sessions and workers are determined by the service tier and compute size (DTUs/eDTUs or vCores). New requests are rejected when session or worker limits are reached, and clients receive an error message. While the number of connections available can be controlled by the application, the number of concurrent workers is often harder to estimate and control. This is especially true during peak load periods when database resource limits are reached and workers pile up due to longer running queries, large blocking chains, or excessive query parallelism.

When encountering high session or worker utilization, mitigation options include:

- Increasing the service tier or compute size of the database or elastic pool. See [Scale single database resources](#) and [Scale elastic pool resources](#).
- Optimizing queries to reduce the resource utilization of each query if the cause of increased worker utilization is due to contention for compute resources. For more information, see [Query Tuning/Hinting](#).

## Resource governance

To enforce resource limits, Azure SQL Database uses a resource governance implementation that is based on SQL Server [Resource Governor](#), modified and extended to run a SQL Server database service in Azure. On each SQL Server instance in the service, there are multiple [resource pools](#) and [workload groups](#), with resource limits set at both pool and group levels to provide a [balanced Database-as-a-Service](#). User workload and internal workloads are classified into separate resource pools and workload groups. User workload on the primary and readable secondary replicas, including geo-replicas, is classified into the `s1oSharedPool1` resource pool and `UserPrimaryGroup.DBId[N]` workload group, where `N` stands for the database ID value. In addition, there are multiple resource pools and workload groups for various internal workloads.

In addition to using Resource Governor to govern resources within the SQL Server process, Azure SQL Database also uses Windows [Job Objects](#) for process level resource governance, and Windows [File Server Resource Manager \(FSRM\)](#) for storage quota management.

Azure SQL Database resource governance is hierarchical in nature. From top to bottom, limits are enforced at the OS level and at the storage volume level using operating system resource governance mechanisms and Resource Governor, then at the resource pool level using Resource Governor, and then at the workload group level using Resource Governor. Resource governance limits in effect for the current database or elastic pool are surfaced in the [sys.dm\\_user\\_db\\_resource\\_governance](#) view.

## Data IO governance

Data IO governance is a process in Azure SQL Database used to limit both read and write physical IO against data files of a database. IOPS limits are set for each service level to minimize the "noisy neighbor" effect, to provide resource allocation fairness in the multi-tenant service, and to stay within the capabilities of the underlying hardware and storage.

For single databases, workload group limits are applied to all storage IO against the database, while resource pool limits apply to all storage IO against all databases on the same SQL Server instance, including the `tempdb` database. For elastic pools, workload group limits apply to each database in the pool, whereas resource pool limit applies to the entire elastic pool, including the `tempdb` database, which is shared among all databases in the pool. In general, resource pool limits may not be achievable by the workload against a database (either single or pooled), because workload group limits are lower than resource pool limits and limit IOPS/throughput sooner. However, pool limits may be reached by the combined workload against multiple databases on the same SQL Server instance.

For example, if a query generates 1000 IOPS without any IO resource governance, but the workload group maximum IOPS limit is set to 900 IOPS, the query will not be able to generate more than 900 IOPS. However, if the resource pool maximum IOPS limit is set to 1500 IOPS, and the total IO from all workload groups associated with the resource pool exceeds 1500 IOPS, then the IO of the same query may be reduced below the workgroup limit of 900 IOPS.

The IOPS and throughput min/max values returned by the [sys.dm\\_user\\_db\\_resource\\_governance](#) view act as limits/caps, not as guarantees. Further, resource governance does not guarantee any specific storage latency. The best achievable latency, IOPS, and throughput for a given user workload depend not only on IO resource governance limits, but also on the mix of IO sizes used, and on the capabilities of the underlying storage. SQL Server uses IOs that vary in size between 512 KB and 4 MB. For the purposes of enforcing IOPS limits, every IO is accounted regardless of its size, with the exception of databases with data files in Azure Storage. In that case, IOs larger than 256 KB are accounted as multiple 256 KB IOs, to align with Azure Storage IO accounting.

For Basic, Standard, and General Purpose databases, which use data files in Azure Storage, the `primary_group_max_io` value may not be achievable if a database does not have enough data files to cumulatively provide this number of IOPS, or if data is not distributed evenly across files, or if the performance tier of underlying blobs limits IOPS/throughput below the resource governance limit. Similarly, with small log IOs generated by frequent transaction commit, the `primary_max_log_rate` value may not be achievable by a workload due to the IOPS limit on the underlying Azure storage blob.

Resource utilization values such as `avg_data_io_percent` and `avg_log_write_percent`, reported in the [sys.dm\\_db\\_resource\\_stats](#), [sys.resource\\_stats](#), and [sys.elastic\\_pool\\_resource\\_stats](#) views, are calculated as percentages of maximum resource governance limits. Therefore, when factors other than resource governance limit IOPS/throughput, it is possible to see IOPS/throughput flattening out and latencies increasing as the workload increases, even though reported resource utilization remains below 100%.

To see read and write IOPS, throughput, and latency per database file, use the [sys.dm\\_io\\_virtual\\_file\\_stats\(\)](#) function. This function surfaces all IO against the database, including background IO that is not accounted towards `avg_data_io_percent`, but uses IOPS and throughput of the underlying storage, and can impact observed storage latency. The function also surfaces additional latency that may be introduced by IO resource governance for reads and writes, in the `io_stall_queued_read_ms` and `io_stall_queued_write_ms` columns respectively.

## Transaction log rate governance

Transaction log rate governance is a process in Azure SQL Database used to limit high ingestion rates for workloads such as bulk insert, SELECT INTO, and index builds. These limits are tracked and enforced at the subsecond level to the rate of log record generation, limiting throughput regardless of how many IOs may be issued against data files. Transaction log generation rates currently scale linearly up to a point that is hardware-dependent, with the maximum log rate allowed being 96 MB/s with the vCore purchasing model.

### NOTE

The actual physical IOs to transaction log files are not governed or limited.

Log rates are set such that they can be achieved and sustained in a variety of scenarios, while the overall system can maintain its functionality with minimized impact to the user load. Log rate governance ensures that transaction log backups stay within published recoverability SLAs. This governance also prevents an excessive backlog on secondary replicas.

As log records are generated, each operation is evaluated and assessed for whether it should be delayed in order to maintain a maximum desired log rate (MB/s per second). The delays are not added when the log records are flushed to storage, rather log rate governance is applied during log rate generation itself.

The actual log generation rates imposed at run time may also be influenced by feedback mechanisms, temporarily reducing the allowable log rates so the system can stabilize. Log file space management, avoiding running into out of log space conditions and Availability Group replication mechanisms can temporarily decrease the overall system limits.

Log rate governor traffic shaping is surfaced via the following wait types (exposed in the [sys.dm\\_db\\_wait\\_stats](#) DMV):

WAIT TYPE	NOTES
LOG_RATE_GOVERNOR	Database limiting
POOL_LOG_RATE_GOVERNOR	Pool limiting
INSTANCE_LOG_RATE_GOVERNOR	Instance level limiting
HADR_THROTTLE_LOG_RATE_SEND_RECV_QUEUE_SIZE	Feedback control, availability group physical replication in Premium/Business Critical not keeping up
HADR_THROTTLE_LOG_RATE_LOG_SIZE	Feedback control, limiting rates to avoid an out of log space condition

When encountering a log rate limit that is hampering desired scalability, consider the following options:

- Scale up to a higher service level in order to get the maximum 96 MB/s log rate.
- If data being loaded is transient, such as staging data in an ETL process, it can be loaded into tempdb (which is minimally logged).
- For analytic scenarios, load into a clustered columnstore covered table. This reduces the required log rate due to compression. This technique does increase CPU utilization and is only applicable to data sets that benefit from clustered columnstore indexes.

## Next steps

- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).
- For information about DTUs and eDTUs, see [DTUs and eDTUs](#).
- For information about tempdb size limits, see [TempDB in Azure SQL Database](#).

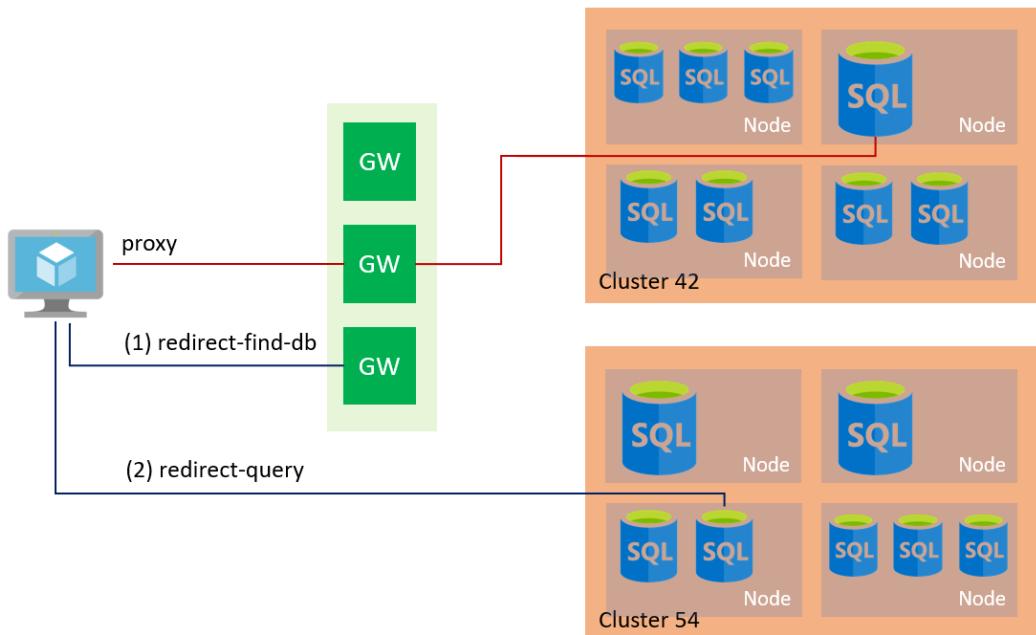
# Azure SQL Connectivity Architecture

1/3/2020 • 6 minutes to read • [Edit Online](#)

This article explains the Azure SQL Database and SQL Data Warehouse connectivity architecture as well as how the different components function to direct traffic to your instance of Azure SQL. These connectivity components function to direct network traffic to the Azure SQL Database or SQL Data Warehouse with clients connecting from within Azure and with clients connecting from outside of Azure. This article also provides script samples to change how connectivity occurs, and the considerations related to changing the default connectivity settings.

## Connectivity architecture

The following diagram provides a high-level overview of the Azure SQL Database connectivity architecture.



The following steps describe how a connection is established to an Azure SQL database:

- Clients connect to the gateway, that has a public IP address and listens on port 1433.
- The gateway, depending on the effective connection policy, redirects or proxies the traffic to the right database cluster.
- Inside the database cluster traffic is forwarded to the appropriate Azure SQL database.

## Connection policy

Azure SQL Database supports the following three options for the connection policy setting of a SQL Database server:

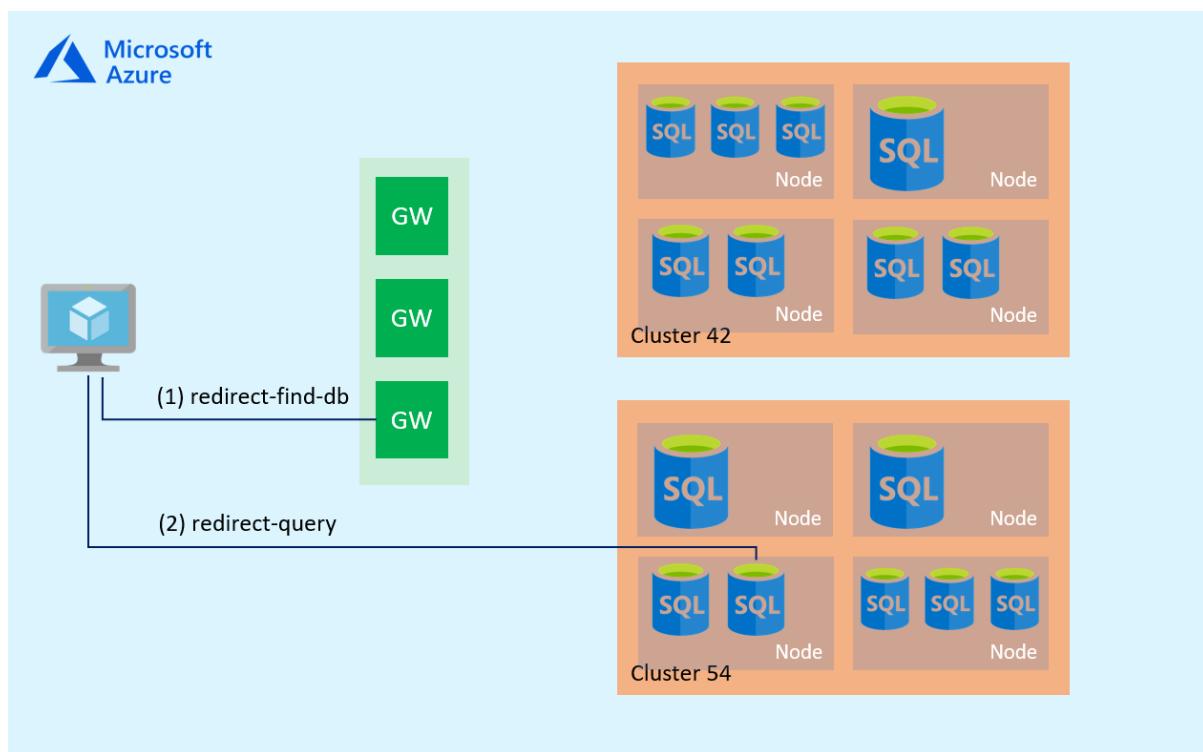
- **Redirect (recommended):** Clients establish connections directly to the node hosting the database, leading to reduced latency and improved throughput. For connections to use this mode clients need to
  - Allow inbound and outbound communication from the client to all Azure IP addresses in the region on ports in the range of 11000-11999.

- Allow inbound and outbound communication from the client to Azure SQL Database gateway IP addresses on port 1433.
- **Proxy:** In this mode, all connections are proxied via the Azure SQL Database gateways, leading to increased latency and reduced throughput. For connections to use this mode clients need to allow inbound and outbound communication from the client to Azure SQL Database gateway IP addresses on port 1433.
- **Default:** This is the connection policy in effect on all servers after creation unless you explicitly alter the connection policy to either **Proxy** or **Redirect**. The default policy is **Redirect** for all client connections originating inside of Azure (e.g. from an Azure Virtual Machine) and **Proxy** for all client connections originating outside (e.g. connections from your local workstation).

We highly recommend the **Redirect** connection policy over the **Proxy** connection policy for the lowest latency and highest throughput. However, you will need to meet the additional requirements for allowing network traffic as outlined above. If the client is an Azure Virtual Machine you can accomplish this using Network Security Groups (NSG) with [service tags](#). If the client is connecting from a workstation on-premises then you may need to work with your network admin to allow network traffic through your corporate firewall.

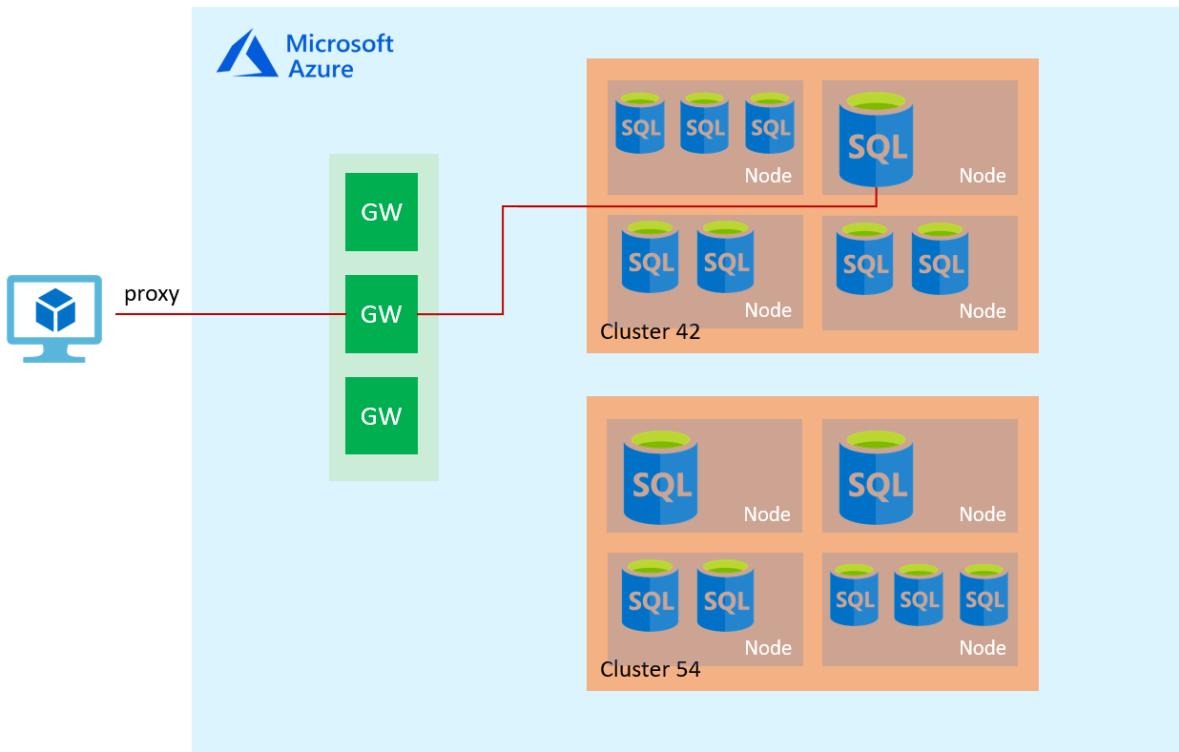
## Connectivity from within Azure

If you are connecting from within Azure your connections have a connection policy of **Redirect** by default. A policy of **Redirect** means that after the TCP session is established to the Azure SQL database, the client session is then redirected to the right database cluster with a change to the destination virtual IP from that of the Azure SQL Database gateway to that of the cluster. Thereafter, all subsequent packets flow directly to the cluster, bypassing the Azure SQL Database gateway. The following diagram illustrates this traffic flow.



## Connectivity from outside of Azure

If you are connecting from outside Azure, your connections have a connection policy of **Proxy** by default. A policy of **Proxy** means that the TCP session is established via the Azure SQL Database gateway and all subsequent packets flow via the gateway. The following diagram illustrates this traffic flow.



#### IMPORTANT

Additionally open ports 14000-14999 to enable [Connecting with DAC](#)

## Azure SQL Database gateway IP addresses

The table below lists the IP Addresses of Gateways by region. To connect to an Azure SQL Database, you need to allow network traffic to & from **all** Gateways for the region.

Details of how traffic shall be migrated to new Gateways in specific regions are in the following article: [Azure SQL Database traffic migration to newer Gateways](#)

REGION NAME	GATEWAY IP ADDRESSES
Australia Central	20.36.105.0
Australia Central2	20.36.113.0
Australia East	13.75.149.87, 40.79.161.1
Australia South East	191.239.192.109, 13.73.109.251
Brazil South	104.41.11.5, 191.233.200.14
Canada Central	40.85.224.249
Canada East	40.86.226.166
Central US	13.67.215.62, 52.182.137.15, 23.99.160.139, 104.208.16.96, 104.208.21.1

REGION NAME	GATEWAY IP ADDRESSES
China East	139.219.130.35
China East 2	40.73.82.1
China North	139.219.15.17
China North 2	40.73.50.0
East Asia	191.234.2.139, 52.175.33.150, 13.75.32.4
East US	40.121.158.30, 40.79.153.12, 191.238.6.43, 40.78.225.32
East US 2	40.79.84.180, 52.177.185.181, 52.167.104.0, 191.239.224.107, 104.208.150.3
France Central	40.79.137.0, 40.79.129.1
Germany Central	51.4.144.100
Germany North East	51.5.144.179
India Central	104.211.96.159
India South	104.211.224.146
India West	104.211.160.80
Japan East	13.78.61.196, 40.79.184.8, 13.78.106.224, 191.237.240.43, 40.79.192.5
Japan West	104.214.148.156, 40.74.100.192, 191.238.68.11, 40.74.97.10
Korea Central	52.231.32.42
Korea South	52.231.200.86
North Central US	23.96.178.199, 23.98.55.75, 52.162.104.33
North Europe	40.113.93.91, 191.235.193.75, 52.138.224.1
South Africa North	102.133.152.0
South Africa West	102.133.24.0
South Central US	13.66.62.124, 23.98.162.75, 104.214.16.32
South East Asia	104.43.15.0, 23.100.117.95, 40.78.232.3
UAE Central	20.37.72.64

REGION NAME	GATEWAY IP ADDRESSES
UAE North	65.52.248.0
UK South	51.140.184.11
UK West	51.141.8.11
West Central US	13.78.145.25
West Europe	40.68.37.158, 191.237.232.75, 104.40.168.105
West US	104.42.238.205, 23.99.34.75, 13.86.216.196
West US 2	13.66.226.202

## Change Azure SQL Database connection policy

To change the Azure SQL Database connection policy for an Azure SQL Database server, use the [conn-policy](#) command.

- If your connection policy is set to `Proxy`, all network packets flow via the Azure SQL Database gateway. For this setting, you need to allow outbound to only the Azure SQL Database gateway IP. Using a setting of `Proxy` has more latency than a setting of `Redirect`.
- If your connection policy is setting `Redirect`, all network packets flow directly to the database cluster. For this setting, you need to allow outbound to multiple IPs.

## Script to change connection settings via PowerShell

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. The following script requires the [Azure PowerShell module](#).

The following PowerShell script shows how to change the connection policy.

```
Get SQL Server ID
$sqlserverid=(Get-AzSqlServer -ServerName sql-server-name -ResourceGroupName sql-server-group).ResourceId

Set URI
$id="$sqlserverid/connectionPolicies/Default"

Get current connection policy
(Get-AzResource -ResourceId $id).Properties.connectionType

Update connection policy
Set-AzResource -ResourceId $id -Properties @{"connectionType" = "Proxy"} -f
```

## Script to change connection settings via Azure CLI

### IMPORTANT

This script requires the [Azure CLI](#).

### Azure CLI in a bash shell

### IMPORTANT

This script requires the [Azure CLI](#).

The following CLI script shows how to change the connection policy in a bash shell.

```
Get SQL Server ID
$sqlserverid=$(az sql server show -n sql-server-name -g sql-server-group --query 'id' -o tsv)

Set URI
$ids="$sqlserverid/connectionPolicies/Default"

Get current connection policy
az resource show --ids $ids

Update connection policy
az resource update --ids $ids --set properties.connectionType=Proxy
```

### Azure CLI from a Windows command prompt

### IMPORTANT

This script requires the [Azure CLI](#).

The following CLI script shows how to change the connection policy from a Windows command prompt (with Azure CLI installed).

```
Get SQL Server ID and set URI
FOR /F "tokens=*" %g IN ('az sql server show --resource-group myResourceGroup-571418053 --name server-538465606 --query "id" -o tsv') do (SET sqlserverid=%g/connectionPolicies/Default)

Get current connection policy
az resource show --ids %sqlserverid%

Update connection policy
az resource update --ids %sqlserverid% --set properties.connectionType=Proxy
```

## Next steps

- For information on how to change the Azure SQL Database connection policy for an Azure SQL Database server, see [conn-policy](#).
- For information about Azure SQL Database connection behavior for clients that use ADO.NET 4.5 or a later version, see [Ports beyond 1433 for ADO.NET 4.5](#).
- For general application development overview information, see [SQL Database Application Development Overview](#).

# Data Sync Agent for Azure SQL Data Sync

11/7/2019 • 9 minutes to read • [Edit Online](#)

Sync data with on-premises SQL Server databases by installing and configuring the Data Sync Agent for Azure SQL Data Sync. For more info about SQL Data Sync, see [Sync data across multiple cloud and on-premises databases with SQL Data Sync](#).

## IMPORTANT

Azure SQL Data Sync does **not** support Azure SQL Database Managed Instance at this time.

## Download and install

To download the Data Sync Agent, go to [SQL Azure Data Sync Agent](#).

### Install silently

To install the Data Sync Agent silently from the command prompt, enter a command similar to the following example. Check the file name of the downloaded .msi file, and provide your own values for the **TARGETDIR** and **SERVICEACCOUNT** arguments.

- If you don't provide a value for **TARGETDIR**, the default value is  
`C:\Program Files (x86)\Microsoft SQL Data Sync 2.0`.
- If you provide `LocalSystem` as the value of **SERVICEACCOUNT**, use SQL Server authentication when you configure the agent to connect to the on-premises SQL Server.
- If you provide a domain user account or a local user account as the value of **SERVICEACCOUNT**, you also have to provide the password with the **SERVICEPASSWORD** argument. For example,  
`SERVICEACCOUNT=<domain>\<user>" SERVICEPASSWORD=<password>"`.

```
msiexec /i "SQLDataSyncAgent-2.0-x86-ENU.msi" TARGETDIR="C:\Program Files (x86)\Microsoft SQL Data Sync 2.0"
SERVICEACCOUNT="LocalSystem" /qn
```

## Sync data with SQL Server on-premises

To configure the Data Sync Agent so you can sync data with one or more on-premises SQL Server databases, see [Add an on-premises SQL Server database](#).

## Data Sync Agent FAQ

### Why do I need a client agent

The SQL Data Sync service communicates with SQL Server databases via the client agent. This security feature prevents direct communication with databases behind a firewall. When the SQL Data Sync service communicates with the agent, it does so using encrypted connections and a unique token or *agent key*. The SQL Server databases authenticate the agent using the connection string and agent key. This design provides a high level of security for your data.

### How many instances of the local agent UI can be run

Only one instance of the UI can be run.

## How can I change my service account

After you install a client agent, the only way to change the service account is to uninstall it and install a new client agent with the new service account.

## How do I change my agent key

An agent key can only be used once by an agent. It cannot be reused when you remove then reinstall a new agent, nor can it be used by multiple agents. If you need to create a new key for an existing agent, you must be sure that the same key is recorded with the client agent and with the SQL Data Sync service.

## How do I retire a client agent

To immediately invalidate or retire an agent, regenerate its key in the portal but do not submit it in the Agent UI. Regenerating a key invalidates the previous key irrespective if the corresponding agent is online or offline.

## How do I move a client agent to another computer

If you want to run the local agent from a different computer than it is currently on, do the following things:

1. Install the agent on desired computer.
2. Log in to the SQL Data Sync portal and regenerate an agent key for the new agent.
3. Use the new agent's UI to submit the new agent key.
4. Wait while the client agent downloads the list of on-premises databases that were registered earlier.
5. Provide database credentials for all databases that display as unreachable. These databases must be reachable from the new computer on which the agent is installed.

## Troubleshoot Data Sync Agent issues

- [The client agent install, uninstall, or repair fails](#)
- [The client agent doesn't work after I cancel the uninstall](#)
- [My database isn't listed in the agent list](#)
- [Client agent doesn't start \(Error 1069\)](#)
- [I can't submit the agent key](#)
- [The client agent can't be deleted from the portal if its associated on-premises database is unreachable](#)
- [Local Sync Agent app can't connect to the local sync service](#)

### The client agent install, uninstall, or repair fails

- **Cause.** Many scenarios might cause this failure. To determine the specific cause for this failure, look at the logs.
- **Resolution.** To find the specific cause of the failure, generate and look at the Windows Installer logs. You can turn on logging at a command prompt. For example, if the downloaded installation file is `SQLDataSyncAgent-2.0-x86-ENU.msi`, generate and examine log files by using the following command lines:

- For installs: `msiexec.exe /i SQLDataSyncAgent-2.0-x86-ENU.msi /l*v LocalAgentSetup.Log`
- For uninstalls: `msiexec.exe /x SQLDataSyncAgent-2.0-x86-ENU.msi /l*v LocalAgentSetup.Log`

You can also turn on logging for all installations that are performed by Windows Installer. The Microsoft Knowledge Base article [How to enable Windows Installer logging](#) provides a one-click solution to turn on logging for Windows Installer. It also provides the location of the logs.

### The client agent doesn't work after I cancel the uninstall

The client agent doesn't work, even after you cancel its uninstallation.

- **Cause.** This occurs because the SQL Data Sync client agent doesn't store credentials.
- **Resolution.** You can try these two solutions:
  - Use services.msc to reenter the credentials for the client agent.
  - Uninstall this client agent and then install a new one. Download and install the latest client agent from [Download Center](#).

### My database isn't listed in the agent list

When you attempt to add an existing SQL Server database to a sync group, the database doesn't appear in the list of agents.

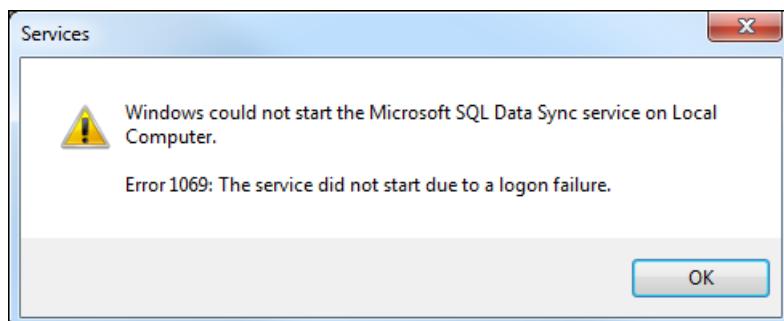
These scenarios might cause this issue:

- **Cause.** The client agent and sync group are in different datacenters.
- **Resolution.** The client agent and the sync group must be in the same datacenter. To set this up, you have two options:
  - Create a new agent in the datacenter where the sync group is located. Then, register the database with that agent.
  - Delete the current sync group. Then, re-create the sync group in the datacenter where the agent is located.
- **Cause.** The client agent's list of databases isn't current.
- **Resolution.** Stop and then restart the client agent service.

The local agent downloads the list of associated databases only on the first submission of the agent key. It doesn't download the list of associated databases on subsequent agent key submissions. Databases that are registered during an agent move don't show up in the original agent instance.

### Client agent doesn't start (Error 1069)

You discover that the agent isn't running on a computer that hosts SQL Server. When you attempt to manually start the agent, you see a dialog box that displays the message, "Error 1069: The service did not start due to a logon failure."

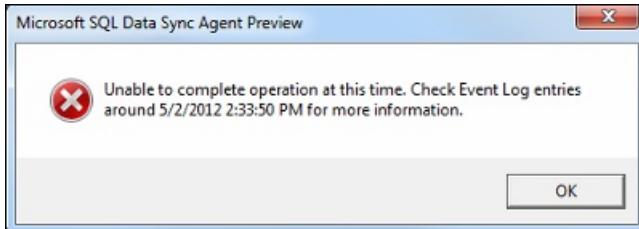


- **Cause.** A likely cause of this error is that the password on the local server has changed since you created the agent and agent password.
- **Resolution.** Update the agent's password to your current server password:
  1. Locate the SQL Data Sync client agent service.
    - a. Select **Start**.
    - b. In the search box, enter **services.msc**.
    - c. In the search results, select **Services**.
    - d. In the **Services** window, scroll to the entry for **SQL Data Sync Agent**.
  2. Right-click **SQL Data Sync Agent**, and then select **Stop**.
  3. Right-click **SQL Data Sync Agent**, and then select **Properties**.

4. On **SQL Data Sync Agent Properties**, select the **Log in** tab.
5. In the **Password** box, enter your password.
6. In the **Confirm Password** box, reenter your password.
7. Select **Apply**, and then select **OK**.
8. In the **Services** window, right-click the **SQL Data Sync Agent** service, and then click **Start**.
9. Close the **Services** window.

### I can't submit the agent key

After you create or re-create a key for an agent, you try to submit the key through the SqlAzureDataSyncAgent application. The submission fails to complete.



- **Prerequisites.** Before you proceed, check the following prerequisites:
  - The SQL Data Sync Windows service is running.
  - The service account for SQL Data Sync Windows service has network access.
  - The outbound 1433 port is open in your local firewall rule.
  - The local ip is added to the server or database firewall rule for the sync metadata database.
- **Cause.** The agent key uniquely identifies each local agent. The key must meet two conditions:
  - The client agent key on the SQL Data Sync server and the local computer must be identical.
  - The client agent key can be used only once.
- **Resolution.** If your agent isn't working, it's because one or both of these conditions are not met. To get your agent to work again:
  1. Generate a new key.
  2. Apply the new key to the agent.

To apply the new key to the agent:

1. In File Explorer, go to your agent installation directory. The default installation directory is C:\Program Files (x86)\Microsoft SQL Data Sync.
2. Double-click the bin subdirectory.
3. Open the SqlAzureDataSyncAgent application.
4. Select **Submit Agent Key**.
5. In the space provided, paste the key from your clipboard.
6. Select **OK**.
7. Close the program.

### The client agent can't be deleted from the portal if its associated on-premises database is unreachable

If a local endpoint (that is, a database) that is registered with a SQL Data Sync client agent becomes unreachable, the client agent can't be deleted.

- **Cause.** The local agent can't be deleted because the unreachable database is still registered with the agent. When you try to delete the agent, the deletion process tries to reach the database, which fails.
- **Resolution.** Use "force delete" to delete the unreachable database.

**NOTE**

If sync metadata tables remain after a "force delete", use `deprovisioningutil.exe` to clean them up.

**Local Sync Agent app can't connect to the local sync service**

- **Resolution.** Try the following steps:

1. Exit the app.
2. Open the Component Services Panel.
  - a. In the search box on the taskbar, enter **services.msc**.
  - b. In the search results, double-click **Services**.
3. Stop the **SQL Data Sync** service.
4. Restart the **SQL Data Sync** service.
5. Reopen the app.

**Run the Data Sync Agent from the command prompt**

You can run the following Data Sync Agent commands from the command prompt:

**Ping the service****Usage**

```
SqlDataSyncAgentCommand.exe -action pingsyncservice
```

**Example**

```
SqlDataSyncAgentCommand.exe -action "pingsyncservice"
```

**Display registered databases****Usage**

```
SqlDataSyncAgentCommand.exe -action displayregistereddatabases
```

**Example**

```
SqlDataSyncAgentCommand.exe -action "displayregistereddatabases"
```

**Submit the agent key****Usage**

```
Usage: SqlDataSyncAgentCommand.exe -action submitagentkey -agentkey [agent key] -username [user name] -password [password]
```

**Example**

```
SqlDataSyncAgentCommand.exe -action submitagentkey -agentkey [agent key generated from portal, PowerShell, or API] -username [user name to sync metadata database] -password [user name to sync metadata database]
```

**Register a database****Usage**

```
SqlDataSyncAgentCommand.exe -action registerdatabase -servername [on-premisesdatabase server name] -
databasename [on-premisesdatabase name] -username [domain\\username] -password [password] -authentication
[sql or windows] -encryption [true or false]
```

## Examples

```
SqlDataSyncAgentCommand.exe -action "registerdatabase" -serverName localhost -databaseName testdb -
authentication sql -username <user name> -password <password> -encryption true
```

```
SqlDataSyncAgentCommand.exe -action "registerdatabase" -serverName localhost -databaseName testdb -
authentication windows -encryption true
```

## Unregister a database

When you use this command to unregister a database, it deprovisions the database completely. If the database participates in other sync groups, this operation breaks the other sync groups.

### Usage

```
SqlDataSyncAgentCommand.exe -action unregisterdatabase -servername [on-premisesdatabase server name] -
databasename [on-premisesdatabase name]
```

### Example

```
SqlDataSyncAgentCommand.exe -action "unregisterdatabase" -serverName localhost -databaseName testdb
```

## Update credentials

### Usage

```
SqlDataSyncAgentCommand.exe -action updatecredential -servername [on-premisesdatabase server name] -
databasename [on-premisesdatabase name] -username [domain\\username] -password [password] -authentication
[sql or windows] -encryption [true or false]
```

## Examples

```
SqlDataSyncAgentCommand.exe -action "updatecredential" -serverName localhost -databaseName testdb -
authentication sql -username <user name> -password <password> -encryption true
```

```
SqlDataSyncAgentCommand.exe -action "updatecredential" -serverName localhost -databaseName testdb -
authentication windows -encryption true
```

## Next steps

For more info about SQL Data Sync, see the following articles:

- Overview - [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- Set up Data Sync
  - In the portal - [Tutorial: Set up SQL Data Sync to sync data between Azure SQL Database and SQL Server on-premises](#)
  - With PowerShell
    - [Use PowerShell to sync between multiple Azure SQL databases](#)
    - [Use PowerShell to sync between an Azure SQL Database and a SQL Server on-premises database](#)
- Best practices - [Best practices for Azure SQL Data Sync](#)

- Monitor - [Monitor SQL Data Sync with Azure Monitor logs](#)
- Troubleshoot - [Troubleshoot issues with Azure SQL Data Sync](#)
- Update the sync schema
  - With Transact-SQL - [Automate the replication of schema changes in Azure SQL Data Sync](#)
  - With PowerShell - [Use PowerShell to update the sync schema in an existing sync group](#)

# Automate the replication of schema changes in Azure SQL Data Sync

11/7/2019 • 8 minutes to read • [Edit Online](#)

SQL Data Sync lets users synchronize data between Azure SQL databases and on-premises SQL Server in one direction or in both directions. One of the current limitations of SQL Data Sync is a lack of support for the replication of schema changes. Every time you change the table schema, you need to apply the changes manually on all endpoints, including the hub and all members, and then update the sync schema.

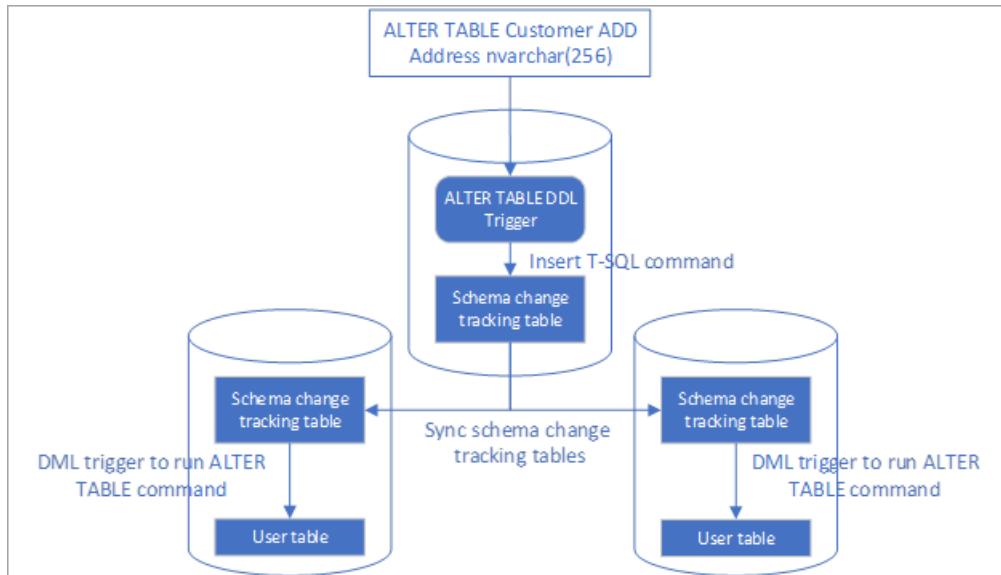
This article introduces a solution to automatically replicate schema changes to all SQL Data Sync endpoints.

1. This solution uses a DDL trigger to track schema changes.
2. The trigger inserts the schema change commands in a tracking table.
3. This tracking table is synced to all endpoints using the Data Sync service.
4. DML triggers after insertion are used to apply the schema changes on the other endpoints.

This article uses `ALTER TABLE` as an example of a schema change, but this solution also works for other types of schema changes.

## IMPORTANT

We recommend that you read this article carefully, especially the sections about [Troubleshooting](#) and [Other considerations](#), before you start to implement automated schema change replication in your sync environment. We also recommend that you read [Sync data across multiple cloud and on-premises databases with SQL Data Sync](#). Some database operations may break the solution described in this article. Additional domain knowledge of SQL Server and Transact-SQL may be required to troubleshoot those issues.



## Set up automated schema change replication

### Create a table to track schema changes

Create a table to track schema changes in all databases in the sync group:

```
CREATE TABLE SchemaChanges (
 ID bigint IDENTITY(1,1) PRIMARY KEY,
 SqlStmt nvarchar(max),
 [Description] nvarchar(max)
)
```

This table has an identity column to track the order of schema changes. You can add more fields to log more information if needed.

### Create a table to track the history of schema changes

On all endpoints, create a table to track the ID of the most recently applied schema change command.

```
CREATE TABLE SchemaChangeHistory (
 LastAppliedId bigint PRIMARY KEY
)
GO

INSERT INTO SchemaChangeHistory VALUES (0)
```

### Create an ALTER TABLE DDL trigger in the database where schema changes are made

Create a DDL trigger for ALTER TABLE operations. You only need to create this trigger in the database where schema changes are made. To avoid conflicts, only allow schema changes in one database in a sync group.

```
CREATE TRIGGER AlterTableDDLTrigger
ON DATABASE
FOR ALTER_TABLE
AS

-- You can add your own logic to filter ALTER TABLE commands instead of replicating all of them.

IF NOT (EVENTDATA().value('/EVENT_INSTANCE/SchemaName')[1], 'nvarchar(512)') like 'DataSync')

 INSERT INTO SchemaChanges (SqlStmt, Description)
 VALUES (EVENTDATA().value('/EVENT_INSTANCE/TSQLCommand/CommandText')[1], 'nvarchar(max)'), 'From DDL
trigger')
```

The trigger inserts a record in the schema change tracking table for each ALTER TABLE command. This example adds a filter to avoid replicating schema changes made under schema **DataSync**, because these are most likely made by the Data Sync service. Add more filters if you only want to replicate certain types of schema changes.

You can also add more triggers to replicate other types of schema changes. For example, create CREATE\_PROCEDURE, ALTER\_PROCEDURE and DROP\_PROCEDURE triggers to replicate changes to stored procedures.

### Create a trigger on other endpoints to apply schema changes during insertion

This trigger executes the schema change command when it is synced to other endpoints. You need to create this trigger on all the endpoints, except the one where schema changes are made (that is, in the database where the DDL trigger `AlterTableDDLTrigger` is created in the previous step).

```

CREATE TRIGGER SchemaChangesTrigger
ON SchemaChanges
AFTER INSERT
AS
DECLARE @lastAppliedId bigint
DECLARE @id bigint
DECLARE @sqlStmt nvarchar(max)
SELECT TOP 1 @lastAppliedId=LastAppliedId FROM SchemaChangeHistory
SELECT TOP 1 @id = id, @SqlStmt = SqlStmt FROM SchemaChanges WHERE id > @lastAppliedId ORDER BY id
IF (@id = @lastAppliedId + 1)
BEGIN
 EXEC sp_executesql @SqlStmt
 UPDATE SchemaChangeHistory SET LastAppliedId = @id
 WHILE (1 = 1)
 BEGIN
 SET @id = @id + 1
 IF exists (SELECT id FROM SchemaChanges WHERE ID = @id)
 BEGIN
 SELECT @sqlStmt = SqlStmt FROM SchemaChanges WHERE ID = @id
 EXEC sp_executesql @SqlStmt
 UPDATE SchemaChangeHistory SET LastAppliedId = @id
 END
 ELSE
 BREAK;
 END
END

```

This trigger runs after the insertion and checks whether the current command should run next. The code logic ensures that no schema change statement is skipped, and all changes are applied even if the insertion is out of order.

### **Sync the schema change tracking table to all endpoints**

You can sync the schema change tracking table to all endpoints using the existing sync group or a new sync group. Make sure the changes in the tracking table can be synced to all endpoints, especially when you're using one-direction sync.

Don't sync the schema change history table, since that table maintains different state on different endpoints.

### **Apply the schema changes in a sync group**

Only schema changes made in the database where the DDL trigger is created are replicated. Schema changes made in other databases are not replicated.

After the schema changes are replicated to all endpoints, you also need to take extra steps to update the sync schema to start or stop syncing the new columns.

#### **Add new columns**

1. Make the schema change.
2. Avoid any data change where the new columns are involved until you've completed the step that creates the trigger.
3. Wait until the schema changes are applied to all endpoints.
4. Refresh the database schema and add the new column to the sync schema.
5. Data in the new column is synced during next sync operation.

#### **Remove columns**

1. Remove the columns from the sync schema. Data Sync stops syncing data in these columns.
2. Make the schema change.

3. Refresh the database schema.

#### Update data types

1. Make the schema change.
2. Wait until the schema changes are applied to all endpoints.
3. Refresh the database schema.
4. If the new and old data types are not fully compatible - for example, if you change from `int` to `bigint` - sync may fail before the steps that create the triggers are completed. Sync succeeds after a retry.

#### Rename columns or tables

Renaming columns or tables makes Data Sync stop working. Create a new table or column, backfill the data, and then delete the old table or column instead of renaming.

#### Other types of schema changes

For other types of schema changes - for example, creating stored procedures or dropping an index- updating the sync schema is not required.

## Troubleshoot automated schema change replication

The replication logic described in this article stops working in some situations- for example, if you made a schema change in an on-premises database which is not supported in Azure SQL Database. In that case, syncing the schema change tracking table fails. You need fix this problem manually:

1. Disable the DDL trigger and avoid any further schema changes until the issue is fixed.
2. In the endpoint database where the issue is happening, disable the AFTER INSERT trigger on the endpoint where the schema change can't be made. This action allows the schema change command to be synced.
3. Trigger sync to sync the schema change tracking table.
4. In the endpoint database where the issue is happening, query the schema change history table to get the ID of last applied schema change command.
5. Query the schema change tracking table to list all the commands with an ID greater than the ID value you retrieved in the previous step.
  - a. Ignore those commands that can't be executed in the endpoint database. You need to deal with the schema inconsistency. Revert the original schema changes if the inconsistency impacts your application.
  - b. Manually apply those commands that should be applied.
6. Update the schema change history table and set the last applied ID to the correct value.
7. Double-check whether the schema is up-to-date.
8. Re-enable the AFTER INSERT trigger disabled in the second step.
9. Re-enable the DDL trigger disabled in the first step.

If you want to clean up the records in the schema change tracking table, use `DELETE` instead of `TRUNCATE`.

Never reseed the identity column in schema change tracking table by using `DBCC CHECKIDENT`. You can create new schema change tracking tables and update the table name in the DDL trigger if reseeding is required.

## Other Considerations

- Database users who configure the hub and member databases need to have enough permission to execute the schema change commands.

- You can add more filters in the DDL trigger to only replicate schema change in selected tables or operations.
- You can only make schema changes in the database where the DDL trigger is created.
- If you are making a change in an on-premises SQL Server database, make sure the schema change is supported in Azure SQL Database.
- If schema changes are made in databases other than the database where the DDL trigger is created, the changes are not replicated. To avoid this issue, you can create DDL triggers to block changes on other endpoints.
- If you need to change the schema of the schema change tracking table, disable the DDL trigger before you make the change, and then manually apply the change to all endpoints. Updating the schema in an AFTER INSERT trigger on the same table does not work.
- Don't reseed the identity column by using DBCC CHECKIDENT.
- Don't use TRUNCATE to clean up data in the schema change tracking table.

## Next steps

For more info about SQL Data Sync, see:

- Overview - [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- Set up Data Sync
  - In the portal - [Tutorial: Set up SQL Data Sync to sync data between Azure SQL Database and SQL Server on-premises](#)
  - With PowerShell
    - [Use PowerShell to sync between multiple Azure SQL databases](#)
    - [Use PowerShell to sync between an Azure SQL Database and a SQL Server on-premises database](#)
- Data Sync Agent - [Data Sync Agent for Azure SQL Data Sync](#)
- Best practices - [Best practices for Azure SQL Data Sync](#)
- Monitor - [Monitor SQL Data Sync with Azure Monitor logs](#)
- Troubleshoot - [Troubleshoot issues with Azure SQL Data Sync](#)
- Update the sync schema
  - With PowerShell - [Use PowerShell to update the sync schema in an existing sync group](#)

## Monitor SQL Data Sync with Azure Monitor logs

11/14/2019 • 7 minutes to read • [Edit Online](#)

To check the SQL Data Sync activity log and detect errors and warnings, you previously had to check SQL Data Sync manually in the Azure portal, or use PowerShell or the REST API. Follow the steps in this article to configure a custom solution that improves the Data Sync monitoring experience. You can customize this solution to fit your scenario.

## NOTE

This article was recently updated to use the term Azure Monitor logs instead of Log Analytics. Log data is still stored in a Log Analytics workspace and is still collected and analyzed by the same Log Analytics service. We are updating the terminology to better reflect the role of [logs in Azure Monitor](#). See [Azure Monitor terminology changes](#) for details.

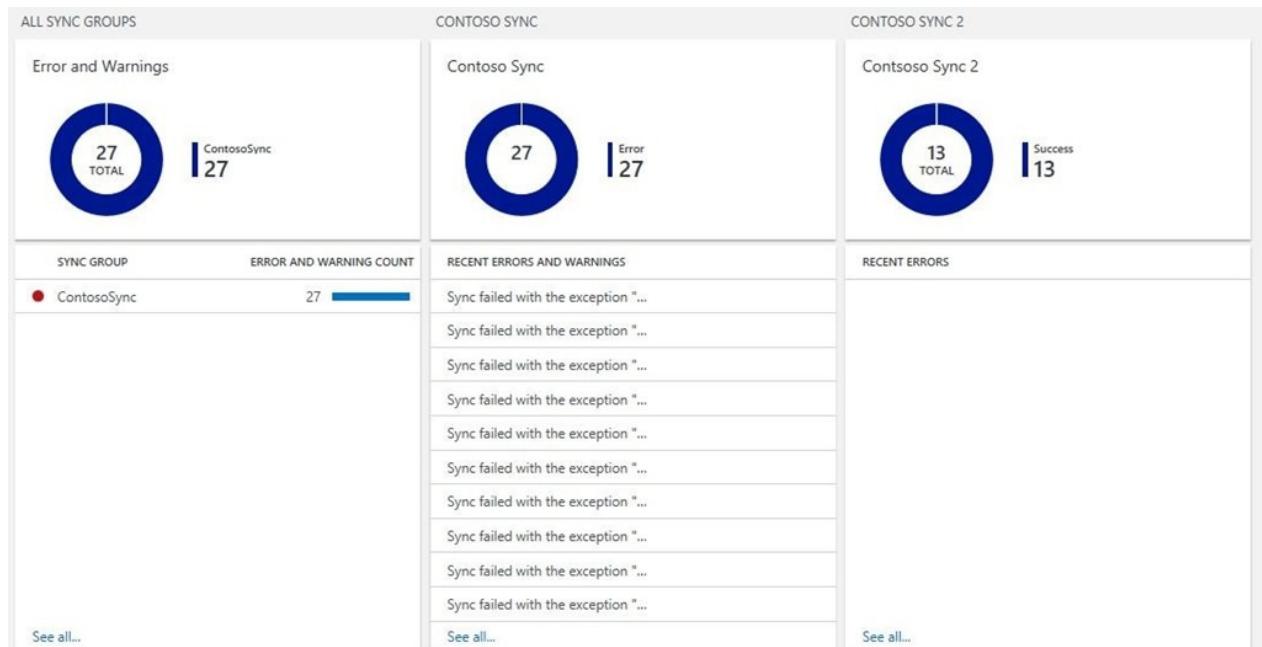
For an overview of SQL Data Sync, see [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#).

## **IMPORTANT**

Azure SQL Data Sync does **not** support Azure SQL Database Managed Instance at this time.

# Monitoring Dashboard for all your Sync Groups

You no longer need to look through the logs of each Sync Group individually to look for issues. You can monitor all your Sync Groups from any of your subscriptions in one place by using a custom Azure Monitor view. This view surfaces the information that matters to SQL Data Sync customers.



## Automated Email notifications

You no longer need to check the log manually in the Azure portal or through PowerShell or the REST API. With [Azure Monitor logs](#), you can create alerts that go directly to the email addresses of the people that need to see

them when an error occurs.

We are notifying you because there are 12 counts of "Data Sync Error"	
NAME	Data Sync Error
SEVERITY	Critical
WORKSPACE NAME	DataSyncOMS
SEARCH INTERVAL START TIME	9/19/2017 10:24:02 PM (UTC)
SEARCH INTERVAL DURATION	60 min
SEARCH QUERY	LogLevel_s=Error
SEARCH RESULTS	<a href="#">12 result(s)</a>

## How do you set up these monitoring features?

Implement a custom Azure Monitor logs monitoring solution for SQL Data Sync in less than an hour by doing the following things:

You need to configure three components:

- A PowerShell runbook to feed SQL Data Sync log data to Azure Monitor logs.
- An Azure Monitor alert for email notifications.
- An Azure Monitor View for monitoring.

### Samples to download

Download the following two samples:

- [Data Sync Log PowerShell Runbook](#)
- [Data Sync Azure Monitor View](#)

### Prerequisites

Make sure you have set up the following things:

- An Azure Automation account
- Log Analytics workspace

## PowerShell Runbook to get SQL Data Sync Log

Use a PowerShell runbook hosted in Azure Automation to pull the SQL Data Sync log data and send it to Azure Monitor logs. A sample script is included. As a prerequisite, you need to have an Azure Automation account. Then you need to create a runbook and schedule it to run.

### Create a runbook

For more info about creating a runbook, see [My first PowerShell runbook](#).

1. Under your Azure Automation account, select the **Runbooks** tab under Process Automation.
2. Select **Add a Runbook** at the top left corner of the Runbooks page.
3. Select **Import an existing Runbook**.
4. Under **Runbook file**, use the given `DataSyncLogPowerShellRunbook` file. Set the **Runbook type** as `PowerShell`. Give the runbook a name.
5. Select **Create**. You now have a runbook.

6. Under your Azure Automation Account, select the **Variables** tab under Shared Resources.
7. Select **Add a variable** on the Variables page. Create a variable to store the last execution time for the runbook. If you have multiple runbooks, you need one variable for each runbook.
8. Set the variable name as `DataSyncLogLastUpdatedTime` and set its Type as DateTime.
9. Select the runbook and click the edit button at the top of the page.
10. Make the changes required for your account and your SQL Data Sync configuration. (For more detailed information, see the sample script.)
  - a. Azure information.
  - b. Sync Group information.
  - c. Azure Monitor logs information. Find this information in Azure Portal | Settings | Connected Sources. For more information about sending data to Azure Monitor logs, see [Send data to Azure Monitor logs with the HTTP Data Collector API \(preview\)](#).
11. Run the runbook in the Test pane. Check to make sure it was successful.

If you have errors, make sure you have the latest PowerShell module installed. You can install the latest PowerShell module in the **Modules Gallery** in your Automation Account.

## 12. Click **Publish**

### Schedule the runbook

To schedule the runbook:

1. Under the runbook, select the **Schedules** tab under Resources.
2. Select **Add a Schedule** on the Schedules page.
3. Select **Link a Schedule to your runbook**.
4. Select **Create a new schedule**.
5. Set **Recurrence** to Recurring and set the interval you want. Use the same interval here, in the script, and in Azure Monitor logs.
6. Select **Create**.

### Check the automation

To monitor whether your automation is running as expected, under **Overview** for your automation account, find the **Job Statistics** view under **Monitoring**. Pin this view to your dashboard for easy viewing. Successful runs of the runbook show as "Completed" and Failed runs show as "Failed."

## Create an Azure Monitor Reader Alert for Email Notifications

To create an alert that uses Azure Monitor logs, do the following things. As a prerequisite, you need to have Azure Monitor logs linked with a Log Analytics workspace.

1. In the Azure portal, select **Log Search**.
2. Create a query to select the errors and warnings by sync group within the interval you selected. For example:

```
DataSyncLog_CL | where LogLevel_s != "Success" | summarize AggregatedValue = count() by bin(TimeGenerated,60m),SyncGroupName_s
```
3. After running the query, select the bell that says **Alert**.

4. Under **Generate alert based on**, select **Metric Measurement**.
  - a. Set the Aggregate Value to **Greater than**.
  - b. After **Greater than**, enter the threshold to elapse before you receive notifications. Transient errors are expected in Data Sync. To reduce noise, set the threshold to 5.
5. Under **Actions**, set **Email notification** to "Yes." Enter the desired email recipients.
6. Click **Save**. The specified recipients now receive email notifications when errors occur.

## Create an Azure Monitor View for Monitoring

This step creates an Azure Monitor view to visually monitor all the specified sync groups. The view includes several components:

- An overview tile, which shows how many errors, successes, and warnings all the sync groups have.
- A tile for all sync groups, which shows the count of errors and warnings per sync group. Groups with no issues don't appear on this tile.
- A tile for each Sync Group, which shows the number of errors, successes, and warnings, and the recent error messages.

To configure the Azure Monitor view, do the following things:

1. On the Log Analytics workspace home page, select the plus on the left to open the **view designer**.
2. Select **Import** on the top bar of the view designer. Then select the "DataSyncLogOMSView" sample file.
3. The sample view is for managing two sync groups. Edit this view to fit your scenario. Click **edit** and make the following changes:
  - a. Create new "Donut & List" objects from the Gallery as needed.
  - b. In each tile, update the queries with your information.
    - a. On each tile, change the TimeStamp\_t interval as desired.
    - b. On the tiles for each Sync Group, update the Sync Group names.
    - c. On each tile, update the title as needed.
4. Click **Save** and the view is ready.

## Cost of this solution

In most cases, this solution is free.

**Azure Automation:** There may be a cost incurred with the Azure Automation account, depending on your usage. The first 500 minutes of job run time per month are free. In most cases, this solution is expected to use less than 500 minutes per month. To avoid charges, schedule the runbook to run at an interval of two hours or more. For more info, see [Automation pricing](#).

**Azure Monitor logs:** There may be a cost associated with Azure Monitor logs depending on your usage. The free tier includes 500 MB of ingested data per day. In most cases, this solution is expected to ingest less than 500 MB per day. To decrease the usage, use the failure-only filtering included in the runbook. If you are using more than 500 MB per day, upgrade to the paid tier to avoid the risk of analytics stopping when the limitation is reached. For more info, see [Azure Monitor logs pricing](#).

## Code samples

Download the code samples described in this article from the following locations:

- [Data Sync Log PowerShell Runbook](#)
- [Data Sync Azure Monitor View](#)

## Next steps

For more info about SQL Data Sync, see:

- Overview - [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- Set up Data Sync
  - In the portal - [Tutorial: Set up SQL Data Sync to sync data between Azure SQL Database and SQL Server on-premises](#)
  - With PowerShell
    - [Use PowerShell to sync between multiple Azure SQL databases](#)
    - [Use PowerShell to sync between an Azure SQL Database and a SQL Server on-premises database](#)
- Data Sync Agent - [Data Sync Agent for Azure SQL Data Sync](#)
- Best practices - [Best practices for Azure SQL Data Sync](#)
- Troubleshoot - [Troubleshoot issues with Azure SQL Data Sync](#)
- Update the sync schema
  - With Transact-SQL - [Automate the replication of schema changes in Azure SQL Data Sync](#)
  - With PowerShell - [Use PowerShell to update the sync schema in an existing sync group](#)

For more info about SQL Database, see:

- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Best practices for SQL Data Sync

1/8/2020 • 9 minutes to read • [Edit Online](#)

This article describes best practices for Azure SQL Data Sync.

For an overview of SQL Data Sync, see [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#).

## IMPORTANT

Azure SQL Data Sync does **not** support Azure SQL Database Managed Instance at this time.

## Security and reliability

### Client agent

- Install the client agent by using the least privileged user account that has network service access.
- Install the client agent on a computer that isn't the on-premises SQL Server computer.
- Don't register an on-premises database with more than one agent.
  - Avoid this even if you are syncing different tables for different sync groups.
  - Registering an on-premises database with multiple client agents poses challenges when you delete one of the sync groups.

### Database accounts with least required privileges

- **For sync setup.** Create/Alter Table; Alter Database; Create Procedure; Select/ Alter Schema; Create User-Defined Type.
- **For ongoing sync.** Select/ Insert/ Update/ Delete on tables that are selected for syncing, and on sync metadata and tracking tables; Execute permission on stored procedures created by the service; Execute permission on user-defined table types.
- **For deprovisioning.** Alter on tables part of sync; Select/ Delete on sync metadata tables; Control on sync tracking tables, stored procedures, and user-defined types.

Azure SQL Database supports only a single set of credentials. To accomplish these tasks within this constraint, consider the following options:

- Change the credentials for different phases (for example, *credentials1* for setup and *credentials2* for ongoing).
- Change the permission of the credentials (that is, change the permission after sync is set up).

## Setup

### Database considerations and constraints

#### SQL Database instance size

When you create a new SQL Database instance, set the maximum size so that it's always larger than the database you deploy. If you don't set the maximum size to larger than the deployed database, sync fails. Although SQL Data Sync doesn't offer automatic growth, you can run the `ALTER DATABASE` command to increase the size of the database after it has been created. Ensure that you stay within the SQL Database instance size limits.

## **IMPORTANT**

SQL Data Sync stores additional metadata with each database. Ensure that you account for this metadata when you calculate space needed. The amount of added overhead is related to the width of the tables (for example, narrow tables require more overhead) and the amount of traffic.

## **Table considerations and constraints**

### **Selecting tables**

You don't have to include all the tables that are in a database in a sync group. The tables that you include in a sync group affect efficiency and costs. Include tables, and the tables they are dependent on, in a sync group only if business needs require it.

### **Primary keys**

Each table in a sync group must have a primary key. The SQL Data Sync service can't sync a table that doesn't have a primary key.

Before using SQL Data Sync in production, test initial and ongoing sync performance.

### **Empty tables provide the best performance**

Empty tables provide the best performance at initialization time. If the target table is empty, Data Sync uses bulk insert to load the data. Otherwise, Data Sync does a row-by-row comparison and insertion to check for conflicts. If performance is not a concern, however, you can set up sync between tables that already contain data.

## **Provisioning destination databases**

SQL Data Sync provides basic database autoprovioning.

This section discusses the limitations of provisioning in SQL Data Sync.

### **Autoprovisioning limitations**

SQL Data Sync has the following limitations for autoprovisioning:

- Select only the columns that are created in the destination table. Any columns that aren't part of the sync group aren't provisioned in the destination tables.
- Indexes are created only for selected columns. If the source table index has columns that aren't part of the sync group, those indexes aren't provisioned in the destination tables.
- Indexes on XML type columns aren't provisioned.
- CHECK constraints aren't provisioned.
- Existing triggers on the source tables aren't provisioned.
- Views and stored procedures aren't created on the destination database.
- ON UPDATE CASCADE and ON DELETE CASCADE actions on foreign key constraints aren't recreated in the destination tables.
- If you have decimal or numeric columns with a precision greater than 28, SQL Data Sync may encounter a conversion overflow issue during sync. We recommend that you limit the precision of decimal or numeric columns to 28 or less.

### **Recommendations**

- Use the SQL Data Sync autoprovisioning capability only when you are trying out the service.
- For production, provision the database schema.

## **Where to locate the hub database**

### **Enterprise-to-cloud scenario**

To minimize latency, keep the hub database close to the greatest concentration of the sync group's database traffic.

### **Cloud-to-cloud scenario**

- When all the databases in a sync group are in one datacenter, the hub should be located in the same datacenter. This configuration reduces latency and the cost of data transfer between datacenters.
- When the databases in a sync group are in multiple datacenters, the hub should be located in the same datacenter as the majority of the databases and database traffic.

#### Mixed scenarios

Apply the preceding guidelines to complex sync group configurations, such as those that are a mix of enterprise-to-cloud and cloud-to-cloud scenarios.

## Sync

### Avoid slow and costly initial sync

In this section, we discuss the initial sync of a sync group. Learn how to help prevent an initial sync from taking longer and being more costly than necessary.

#### How initial sync works

When you create a sync group, start with data in only one database. If you have data in multiple databases, SQL Data Sync treats each row as a conflict that needs to be resolved. This conflict resolution causes the initial sync to go slowly. If you have data in multiple databases, initial sync might take between several days and several months, depending on the database size.

If the databases are in different datacenters, each row must travel between the different datacenters. This increases the cost of an initial sync.

#### Recommendation

If possible, start with data in only one of the sync group's databases.

### Design to avoid sync loops

A sync loop occurs when there are circular references within a sync group. In that scenario, each change in one database is endlessly and circularly replicated through the databases in the sync group.

Ensure that you avoid sync loops, because they cause performance degradation and might significantly increase costs.

### Changes that fail to propagate

#### Reasons that changes fail to propagate

Changes might fail to propagate for one of the following reasons:

- Schema/datatype incompatibility.
- Inserting null in non-nullable columns.
- Violating foreign key constraints.

#### What happens when changes fail to propagate?

- Sync group shows that it's in a **Warning** state.
- Details are listed in the portal UI log viewer.
- If the issue is not resolved for 45 days, the database becomes out of date.

#### NOTE

These changes never propagate. The only way to recover in this scenario is to re-create the sync group.

#### Recommendation

Monitor the sync group and database health regularly through the portal and log interface.

## Maintenance

## Avoid out-of-date databases and sync groups

A sync group or a database in a sync group can become out of date. When a sync group's status is **Out-of-date**, it stops functioning. When a database's status is **Out-of-date**, data might be lost. It's best to avoid this scenario instead of trying to recover from it.

### Avoid out-of-date databases

A database's status is set to **Out-of-date** when it has been offline for 45 days or more. To avoid an **Out-of-date** status on a database, ensure that none of the databases are offline for 45 days or more.

### Avoid out-of-date sync groups

A sync group's status is set to **Out-of-date** when any change in the sync group fails to propagate to the rest of the sync group for 45 days or more. To avoid an **Out-of-date** status on a sync group, regularly check the sync group's history log. Ensure that all conflicts are resolved, and that changes are successfully propagated throughout the sync group databases.

A sync group might fail to apply a change for one of these reasons:

- Schema incompatibility between tables.
- Data incompatibility between tables.
- Inserting a row with a null value in a column that doesn't allow null values.
- Updating a row with a value that violates a foreign key constraint.

To prevent out-of-date sync groups:

- Update the schema to allow the values that are contained in the failed rows.
- Update the foreign key values to include the values that are contained in the failed rows.
- Update the data values in the failed row so they are compatible with the schema or foreign keys in the target database.

## Avoid deprovisioning issues

In some circumstances, unregistering a database with a client agent might cause sync to fail.

### Scenario

1. Sync group A was created by using a SQL Database instance and an on-premises SQL Server database, which is associated with local agent 1.
2. The same on-premises database is registered with local agent 2 (this agent is not associated with any sync group).
3. Unregistering the on-premises database from local agent 2 removes the tracking and meta tables for sync group A for the on-premises database.
4. Sync group A operations fail, with this error: "The current operation could not be completed because the database is not provisioned for sync or you do not have permissions to the sync configuration tables."

### Solution

To avoid this scenario, don't register a database with more than one agent.

To recover from this scenario:

1. Remove the database from each sync group that it belongs to.
2. Add the database back into each sync group that you removed it from.
3. Deploy each affected sync group (this action provisions the database).

## Modifying a sync group

Don't attempt to remove a database from a sync group and then edit the sync group without first deploying one of the changes.

Instead, first remove a database from a sync group. Then, deploy the change and wait for deprovisioning to finish.

When deprovisioning is finished, you can edit the sync group and deploy the changes.

If you attempt to remove a database and then edit a sync group without first deploying one of the changes, one or the other operation fails. The portal interface might become inconsistent. If this happens, refresh the page to restore the correct state.

### Avoid schema refresh timeout

If you have a complex schema to sync, you may encounter an "operation timeout" during a schema refresh if the sync metadata database has a lower SKU (example: basic).

#### Solution

To mitigate this issue, please scale up your sync metadata database to have a higher SKU, such as S3.

## Next steps

For more information about SQL Data Sync, see:

- Overview - [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- Set up Data Sync
  - In the portal - [Tutorial: Set up SQL Data Sync to sync data between Azure SQL Database and SQL Server on-premises](#)
  - With PowerShell
    - [Use PowerShell to sync between multiple Azure SQL databases](#)
    - [Use PowerShell to sync between an Azure SQL Database and a SQL Server on-premises database](#)
- Data Sync Agent - [Data Sync Agent for Azure SQL Data Sync](#)
- Monitor - [Monitor SQL Data Sync with Azure Monitor logs](#)
- Troubleshoot - [Troubleshoot issues with Azure SQL Data Sync](#)
- Update the sync schema
  - With Transact-SQL - [Automate the replication of schema changes in Azure SQL Data Sync](#)
  - With PowerShell - [Use PowerShell to update the sync schema in an existing sync group](#)

For more information about SQL Database, see:

- [SQL Database overview](#)
- [Database lifecycle management](#)

# Troubleshoot issues with SQL Data Sync

11/7/2019 • 10 minutes to read • [Edit Online](#)

This article describes how to troubleshoot known issues with Azure SQL Data Sync. If there is a resolution for an issue, it's provided here.

For an overview of SQL Data Sync, see [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#).

## IMPORTANT

Azure SQL Data Sync does **not** support Azure SQL Database Managed Instance at this time.

## Sync issues

- [Sync fails in the portal UI for on-premises databases that are associated with the client agent](#)
- [My sync group is stuck in the processing state](#)
- [I see erroneous data in my tables](#)
- [I see inconsistent primary key data after a successful sync](#)
- [I see a significant degradation in performance](#)
- [I see this message: "Cannot insert the value NULL into the column <column>. Column does not allow nulls." What does this mean, and how can I fix it?](#)
- [How does Data Sync handle circular references? That is, when the same data is synced in multiple sync groups, and keeps changing as a result?](#)

### **Sync fails in the portal UI for on-premises databases that are associated with the client agent**

Sync fails in the SQL Data Sync portal UI for on-premises databases that are associated with the client agent. On the local computer that's running the agent, you see System.IO.IOException errors in the Event Log. The errors say that the disk has insufficient space.

- **Cause.** The drive has insufficient space.
- **Resolution.** Create more space on the drive on which the %TEMP% directory is located.

### **My sync group is stuck in the processing state**

A sync group in SQL Data Sync has been in the processing state for a long time. It doesn't respond to the **stop** command, and the logs show no new entries.

Any of the following conditions might result in a sync group being stuck in the processing state:

- **Cause.** The client agent is offline
- **Resolution.** Be sure that the client agent is online and then try again.
- **Cause.** The client agent is uninstalled or missing.
- **Resolution.** If the client agent is uninstalled or otherwise missing:
  1. Remove the agent XML file from the SQL Data Sync installation folder, if the file exists.

2. Install the agent on an on-premises computer (it can be the same or a different computer). Then, submit the agent key that's generated in the portal for the agent that's showing as offline.

- **Cause.** The SQL Data Sync service is stopped.
- **Resolution.** Restart the SQL Data Sync service.

1. In the **Start** menu, search for **Services**.
2. In the search results, select **Services**.
3. Find the **SQL Data Sync** service.
4. If the service status is **Stopped**, right-click the service name, and then select **Start**.

#### NOTE

If the preceding information doesn't move your sync group out of the processing state, Microsoft Support can reset the status of your sync group. To have your sync group status reset, in the [Azure SQL Database forum](#), create a post. In the post, include your subscription ID and the sync group ID for the group that needs to be reset. A Microsoft Support engineer will respond to your post, and will let you know when the status has been reset.

### I see erroneous data in my tables

If tables that have the same name but which are from different database schemas are included in a sync, you see erroneous data in the tables after the sync.

- **Cause.** The SQL Data Sync provisioning process uses the same tracking tables for tables that have the same name but which are in different schemas. Because of this, changes from both tables are reflected in the same tracking table. This causes erroneous data changes during sync.
- **Resolution.** Ensure that the names of tables that are involved in a sync are different, even if the tables belong to different schemas in a database.

### I see inconsistent primary key data after a successful sync

A sync is reported as successful, and the log shows no failed or skipped rows, but you observe that primary key data is inconsistent among the databases in the sync group.

- **Cause.** This result is by design. Changes in any primary key column result in inconsistent data in the rows where the primary key was changed.
- **Resolution.** To prevent this issue, ensure that no data in a primary key column is changed. To fix this issue after it has occurred, delete the row that has inconsistent data from all endpoints in the sync group. Then, reinsert the row.

### I see a significant degradation in performance

Your performance degrades significantly, possibly to the point where you can't even open the Data Sync UI.

- **Cause.** The most likely cause is a sync loop. A sync loop occurs when a sync by sync group A triggers a sync by sync group B, which then triggers a sync by sync group A. The actual situation might be more complex, and it might involve more than two sync groups in the loop. The issue is that there is a circular triggering of syncing that's caused by sync groups overlapping one another.
- **Resolution.** The best fix is prevention. Ensure that you don't have circular references in your sync groups. Any row that is synced by one sync group can't be synced by another sync group.

### I see this message: "Cannot insert the value NULL into the column <column>. Column does not allow nulls." What does this mean, and how can I fix it?

This error message indicates that one of the two following issues has occurred:

- A table doesn't have a primary key. To fix this issue, add a primary key to all the tables that you're syncing.

- There's a WHERE clause in your CREATE INDEX statement. Data Sync doesn't handle this condition. To fix this issue, remove the WHERE clause or manually make the changes to all databases.

**How does Data Sync handle circular references? That is, when the same data is synced in multiple sync groups, and keeps changing as a result?**

Data Sync doesn't handle circular references. Be sure to avoid them.

## Client agent issues

To troubleshoot issues with the client agent, see [Troubleshoot Data Sync Agent issues](#).

## Setup and maintenance issues

- [I get a "disk out of space" message](#)
- [I can't delete my sync group](#)
- [I can't unregister an on-premises SQL Server database](#)
- [I don't have sufficient privileges to start system services](#)
- [A database has an "Out-of-Date" status](#)
- [A sync group has an "Out-of-Date" status](#)
- [A sync group can't be deleted within three minutes of uninstalling or stopping the agent](#)
- [What happens when I restore a lost or corrupted database?](#)

### I get a "disk out of space" message

- **Cause.** The "disk out of space" message might appear if leftover files need to be deleted. This might be caused by antivirus software, or files are open when delete operations are attempted.
- **Resolution.** Manually delete the sync files that are in the %temp% folder (`del \*sync\* /s`). Then, delete the subdirectories in the %temp% folder.

#### IMPORTANT

Don't delete any files while sync is in progress.

### I can't delete my sync group

Your attempt to delete a sync group fails. Any of the following scenarios might result in failure to delete a sync group:

- **Cause.** The client agent is offline.
- **Resolution.** Ensure that the client agent is online and then try again.
- **Cause.** The client agent is uninstalled or missing.
- **Resolution.** If the client agent is uninstalled or otherwise missing:
  - a. Remove the agent XML file from the SQL Data Sync installation folder, if the file exists.
  - b. Install the agent on an on-premises computer (it can be the same or a different computer). Then, submit the agent key that's generated in the portal for the agent that's showing as offline.
- **Cause.** A database is offline.
- **Resolution.** Ensure that your SQL databases and SQL Server databases are all online.

- **Cause.** The sync group is provisioning or syncing.
- **Resolution.** Wait until the provisioning or sync process finishes and then retry deleting the sync group.

### I can't unregister an on-premises SQL Server database

- **Cause.** Most likely, you are trying to unregister a database that has already been deleted.
- **Resolution.** To unregister an on-premises SQL Server database, select the database and then select **Force Delete**.

If this operation fails to remove the database from the sync group:

1. Stop and then restart the client agent host service:
  - a. Select the **Start** menu.
  - b. In the search box, enter **services.msc**.
  - c. In the **Programs** section of the search results pane, double-click **Services**.
  - d. Right-click the **SQL Data Sync** service.
  - e. If the service is running, stop it.
  - f. Right-click the service, and then select **Start**.
  - g. Check whether the database is still registered. If it is no longer registered, you're done. Otherwise, proceed with the next step.
2. Open the client agent app (**SqlAzureDataSyncAgent**).
3. Select **Edit Credentials**, and then enter the credentials for the database.
4. Proceed with unregistration.

### I don't have sufficient privileges to start system services

- **Cause.** This error occurs in two situations:
  - The user name and/or the password are incorrect.
  - The specified user account doesn't have sufficient privileges to log on as a service.
- **Resolution.** Grant log-on-as-a-service credentials to the user account:

1. Go to **Start > Control Panel > Administrative Tools > Local Security Policy > Local Policy > User Rights Management**.
2. Select **Log on as a service**.
3. In the **Properties** dialog box, add the user account.
4. Select **Apply**, and then select **OK**.
5. Close all windows.

### A database has an "Out-of-Date" status

- **Cause.** SQL Data Sync removes databases that have been offline from the service for 45 days or more (as counted from the time the database went offline). If a database is offline for 45 days or more and then comes back online, its status is **Out-of-Date**.
- **Resolution.** You can avoid an **Out-of-Date** status by ensuring that none of your databases go offline for 45 days or more.

If a database's status is **Out-of-Date**:

1. Remove the database that has an **Out-of-Date** status from the sync group.
2. Add the database back in to the sync group.

#### **WARNING**

You lose all changes made to this database while it was offline.

## A sync group has an "Out-of-Date" status

- **Cause.** If one or more changes fail to apply for the whole retention period of 45 days, a sync group can become outdated.
- **Resolution.** To avoid an **Out-of-Date** status for a sync group, examine the results of your sync jobs in the history viewer on a regular basis. Investigate and resolve any changes that fail to apply.

If a sync group's status is **Out-of-Date**, delete the sync group and then re-create it.

## A sync group can't be deleted within three minutes of uninstalling or stopping the agent

You can't delete a sync group within three minutes of uninstalling or stopping the associated SQL Data Sync client agent.

- **Resolution.**

1. Remove a sync group while the associated sync agents are online (recommended).
2. If the agent is offline but is installed, bring it online on the on-premises computer. Wait for the status of the agent to appear as **Online** in the SQL Data Sync portal. Then, remove the sync group.
3. If the agent is offline because it was uninstalled:
  - a. Remove the agent XML file from the SQL Data Sync installation folder, if the file exists.
  - b. Install the agent on an on-premises computer (it can be the same or a different computer). Then, submit the agent key that's generated in the portal for the agent that's showing as offline.
  - c. Try to delete the sync group.

## What happens when I restore a lost or corrupted database?

If you restore a lost or corrupted database from a backup, there might be a non-convergence of data in the sync groups to which the database belongs.

## Next steps

For more information about SQL Data Sync, see:

- Overview - [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- Set up Data Sync
  - In the portal - [Tutorial: Set up SQL Data Sync to sync data between Azure SQL Database and SQL Server on-premises](#)
  - With PowerShell
    - [Use PowerShell to sync between multiple Azure SQL databases](#)
    - [Use PowerShell to sync between an Azure SQL Database and a SQL Server on-premises database](#)
- Data Sync Agent - [Data Sync Agent for Azure SQL Data Sync](#)
- Best practices - [Best practices for Azure SQL Data Sync](#)
- Monitor - [Monitor SQL Data Sync with Azure Monitor logs](#)
- Update the sync schema
  - With Transact-SQL - [Automate the replication of schema changes in Azure SQL Data Sync](#)
  - With PowerShell - [Use PowerShell to update the sync schema in an existing sync group](#)

For more information about SQL Database, see:

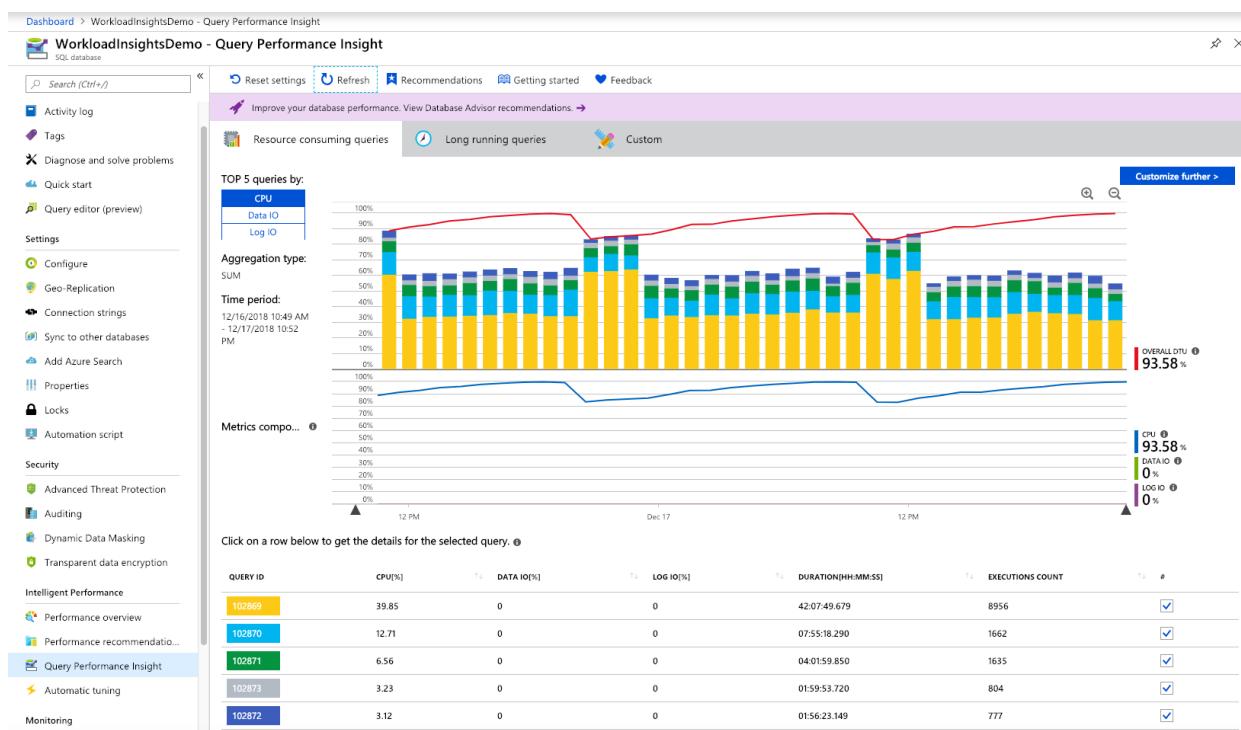
- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Query Performance Insight for Azure SQL Database

11/7/2019 • 10 minutes to read • [Edit Online](#)

Managing and tuning the performance of relational databases takes expertise and time. Query Performance Insight is a part of the Azure SQL Database intelligent performance product line. It helps you spend less time troubleshooting database performance by providing:

- Deeper insight into your databases resource (DTU) consumption.
- Details on top database queries by CPU, duration, and execution count (potential tuning candidates for performance improvements).
- The ability to drill down into details of a query, to view the query text and history of resource utilization.
- Annotations that show performance recommendations from [SQL Database Advisor](#).



## TIP

For basic performance monitoring with Azure SQL Database, we recommend Query Performance Insight. Note the product limitations published in this article. For advanced monitoring of database performance at scale, we recommend [Azure SQL Analytics](#). It has built-in intelligence for automated performance troubleshooting. To automatically tune some of the most common database performance issues, we recommend [Automatic Tuning](#).

## Prerequisites

Query Performance Insight requires that [Query Store](#) is active on your database. It's automatically enabled for all Azure SQL databases by default. If Query Store is not running, the Azure portal will prompt you to enable it.

## NOTE

If the "Query Store is not properly configured on this database" message appears in the portal, see [Optimizing the Query Store configuration](#).

# Permissions

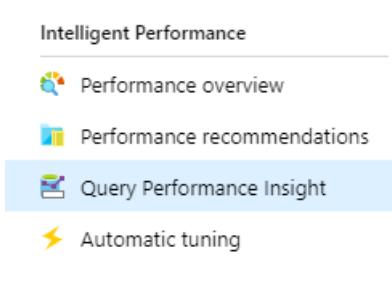
You need the following [role-based access control](#) permissions to use Query Performance Insight:

- **Reader, Owner, Contributor, SQL DB Contributor, or SQL Server Contributor** permissions are required to view the top resource-consuming queries and charts.
- **Owner, Contributor, SQL DB Contributor, or SQL Server Contributor** permissions are required to view query text.

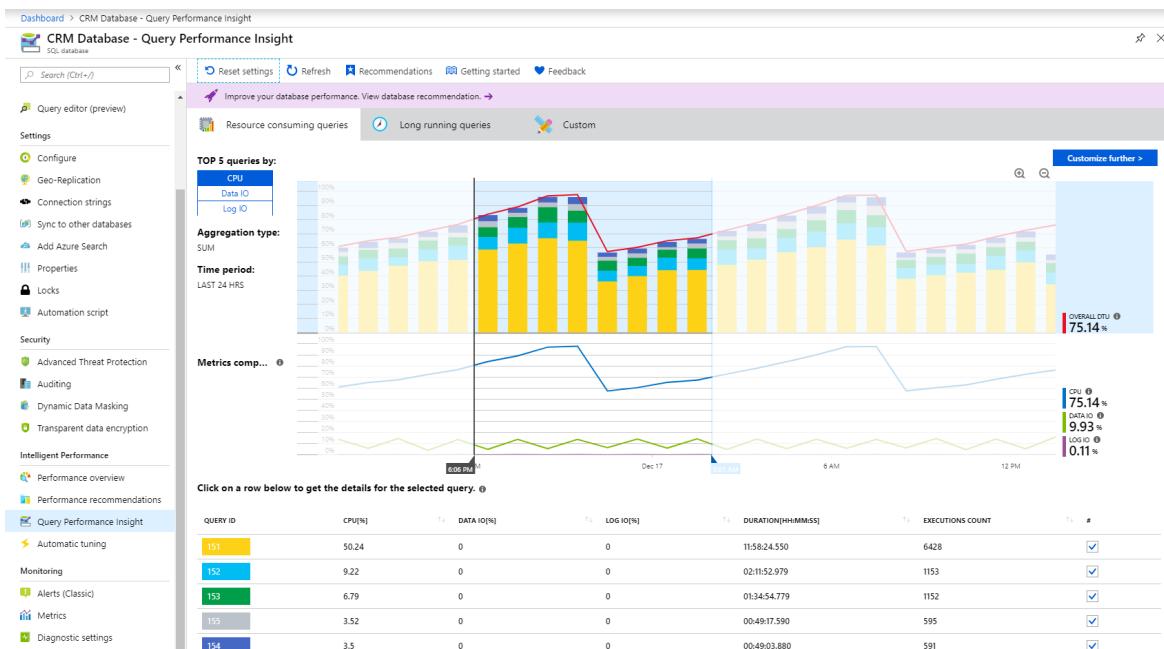
## Use Query Performance Insight

Query Performance Insight is easy to use:

1. Open the [Azure portal](#) and find a database that you want to examine.
2. From the left-side menu, open **Intelligent Performance > Query Performance Insight**.



3. On the first tab, review the list of top resource-consuming queries.
4. Select an individual query to view its details.
5. Open **Intelligent Performance > Performance recommendations** and check if any performance recommendations are available. For more information on built-in performance recommendations, see [SQL Database Advisor](#).
6. Use sliders or zoom icons to change the observed interval.



## NOTE

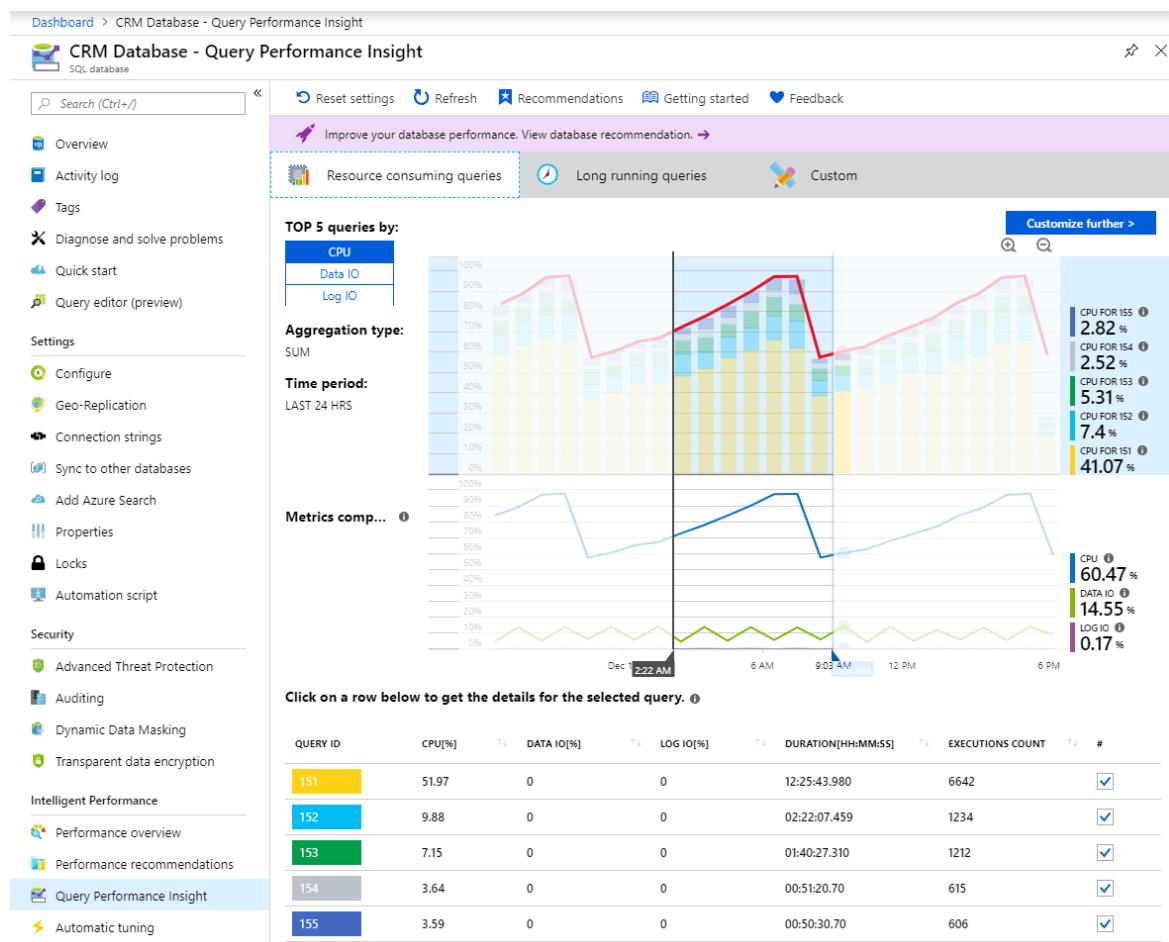
For SQL Database to render the information in Query Performance Insight, Query Store needs to capture a couple hours of data. If the database has no activity or if Query Store was not active during a certain period, the charts will be empty when Query Performance Insight displays that time range. You can enable Query Store at any time if it's not running. For more information, see [Best practices with Query Store](#).

## Review top CPU-consuming queries

By default, Query Performance Insight shows the top five CPU-consuming queries when you first open it.

1. Select or clear individual queries to include or exclude them from the chart by using check boxes.

The top line shows overall DTU percentage for the database. The bars show CPU percentage that the selected queries consumed during the selected interval. For example, if **Past week** is selected, each bar represents a single day.



## IMPORTANT

The DTU line shown is aggregated to a maximum consumption value in one-hour periods. It's meant for a high-level comparison only with query execution statistics. In some cases, DTU utilization might seem too high compared to executed queries, but this might not be the case.

For example, if a query maxed out DTU to 100% for a few minutes only, the DTU line in Query Performance Insight will show the entire hour of consumption as 100% (the consequence of the maximum aggregated value).

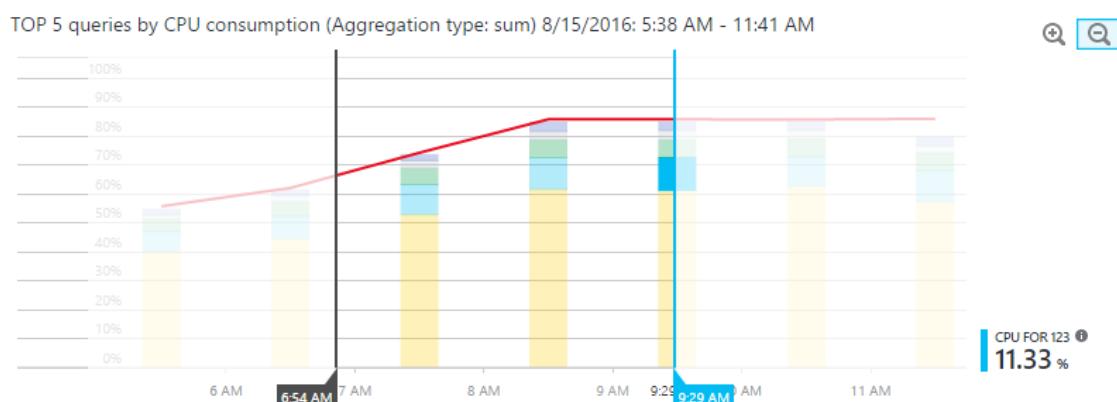
For a finer comparison (up to one minute), consider creating a custom DTU utilization chart:

1. In the Azure portal, select **Azure SQL Database > Monitoring**.
2. Select **Metrics**.
3. Select **+Add chart**.
4. Select the DTU percentage on the chart.
5. In addition, select **Last 24 hours** on the upper-left menu and change it to one minute.

Use the custom DTU chart with a finer level of details to compare with the query execution chart.

The bottom grid shows aggregated information for the visible queries:

- Query ID, which is a unique identifier for the query in the database.
  - CPU per query during an observable interval, which depends on the aggregation function.
  - Duration per query, which also depends on the aggregation function.
  - Total number of executions for a specific query.
2. If your data becomes stale, select the **Refresh** button.
  3. Use sliders and zoom buttons to change the observation interval and investigate consumption spikes:



4. Optionally, you can select the **Custom** tab to customize the view for:

- Metric (CPU, duration, execution count).
- Time interval (last 24 hours, past week, or past month).
- Number of queries.
- Aggregation function.

The screenshot shows the 'Resource consuming queries' section of the Azure portal. It includes tabs for 'Resource consuming queries' and 'Custom'. Below are dropdown menus for Metric type (CPU), Time period (Past week), Number of queries (20), and Aggregation type (max). A 'Go >' button is at the bottom right.

5. Select the **Go >** button to see the customized view.

## IMPORTANT

Query Performance Insight is limited to displaying the top 5-20 consuming queries, depending on your selection. Your database can run many more queries beyond the top ones shown, and these queries will not be included on the chart.

There might exist a database workload type in which lots of smaller queries, beyond the top ones shown, run frequently and use the majority of DTU. These queries don't appear on the performance chart.

For example, a query might have consumed a substantial amount of DTU for a while, although its total consumption in the observed period is less than the other top-consuming queries. In such a case, resource utilization of this query would not appear on the chart.

If you need to understand top query executions beyond the limitations of Query Performance Insight, consider using [Azure SQL Analytics](#) for advanced database performance monitoring and troubleshooting.

## View individual query details

To view query details:

1. Select any query in the list of top queries.

Click on a row below to get the details for the selected query.

QUERY ID	CPU[%]	DURATION[HH:MM:SS]	EXECUTIONS COUNT	#
122	1.27	00:17:31.660	427	<input checked="" type="checkbox"/>
123	0.23	00:03:10.680	77	<input checked="" type="checkbox"/>
124	0.16	00:01:55.460	95	<input checked="" type="checkbox"/>
126	0.09	00:01:12.310	57	<input checked="" type="checkbox"/>

A detailed view opens. It shows the CPU consumption, duration, and execution count over time.

2. Select the chart features for details.

- The top chart shows a line with the overall database DTU percentage. The bars are the CPU percentage that the selected query consumed.
- The second chart shows the total duration of the selected query.
- The bottom chart shows the total number of executions by the selected query.

## Query details

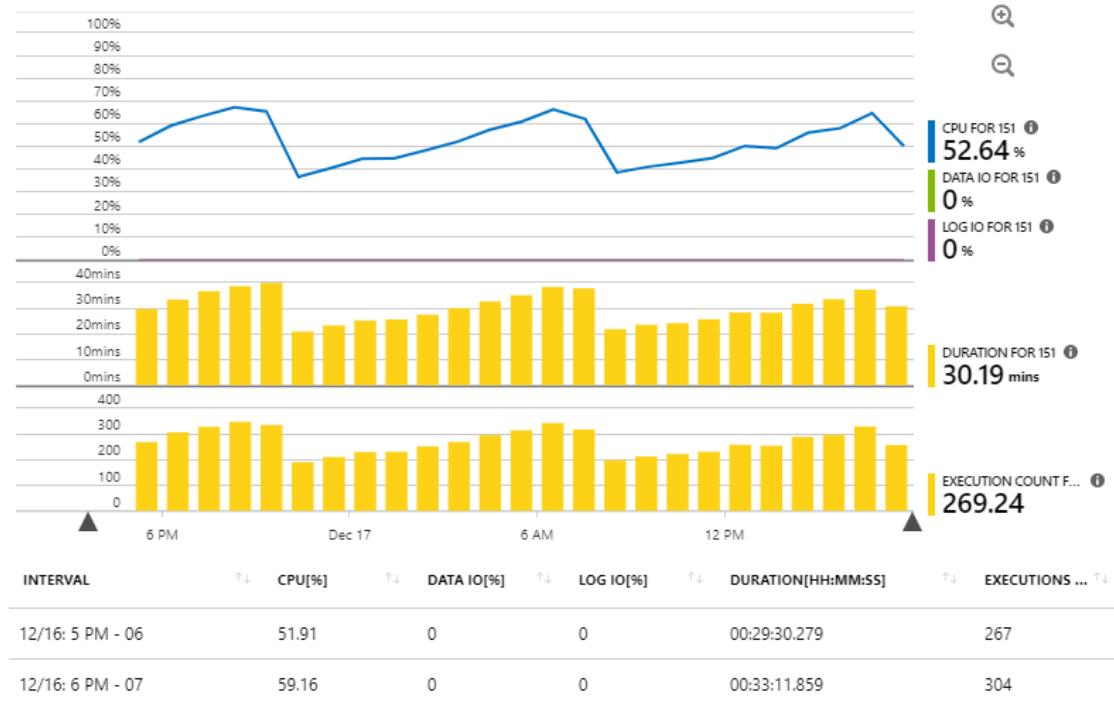
CRM Database - Query ID 151

[Settings](#) [Refresh](#) [Recommendations](#) [Query Text](#)

### Query ID 151:

```
1 SELECT TOP 77 /* asda */c1.col0, c2.col5 FROM Table_0 c1 join Table_0 c2 on square
(c1.col5)<>tan(c2.col7) join Table_0 c3 on sin(c1.col1)*cos(c3.col5)<power(c2.col7,-1)
ORDER BY NEWID()
```

### Details of Query ID 151 (Aggregation type: sum) Last 24 hrs



3. Optionally, use sliders, use zoom buttons, or select **Settings** to customize how query data is displayed, or to pick a different time range.

#### IMPORTANT

Query Performance Insight does not capture any DDL queries. In some cases, it might not capture all ad hoc queries.

## Review top queries per duration

Two metrics in Query Performance Insight can help you find potential bottlenecks: duration and execution count.

Long-running queries have the greatest potential for locking resources longer, blocking other users, and limiting scalability. They're also the best candidates for optimization.

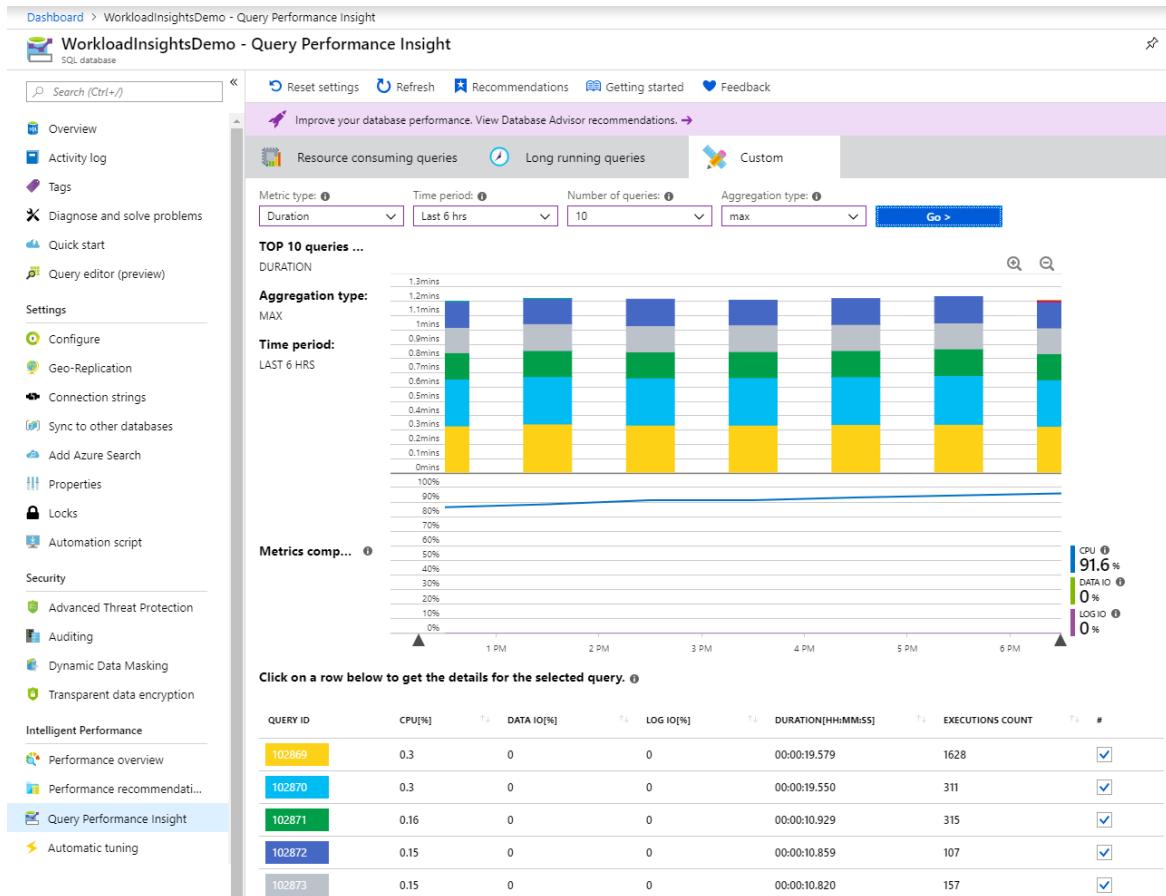
To identify long-running queries:

1. Open the **Custom** tab in Query Performance Insight for the selected database.
2. Change the metrics to **duration**.

3. Select the number of queries and the observation interval.

4. Select the aggregation function:

- **Sum** adds up all query execution time for the whole observation interval.
- **Max** finds queries in which execution time was maximum for the whole observation interval.
- **Avg** finds the average execution time of all query executions and shows you the top ones for these averages.



5. Select the **Go >** button to see the customized view.

#### IMPORTANT

Adjusting the query view does not update the DTU line. The DTU line always shows the maximum consumption value for the interval.

To understand database DTU consumption with more detail (up to one minute), consider creating a custom chart in the Azure portal:

1. Select **Azure SQL Database > Monitoring**.
2. Select **Metrics**.
3. Select **+Add chart**.
4. Select the DTU percentage on the chart.
5. In addition, select **Last 24 hours** on the upper-left menu and change it to one minute.

We recommend that you use the custom DTU chart to compare with the query performance chart.

## Review top queries per execution count

A user application that uses the database might get slow, even though a high number of executions might not be affecting the database itself and resources usage is low.

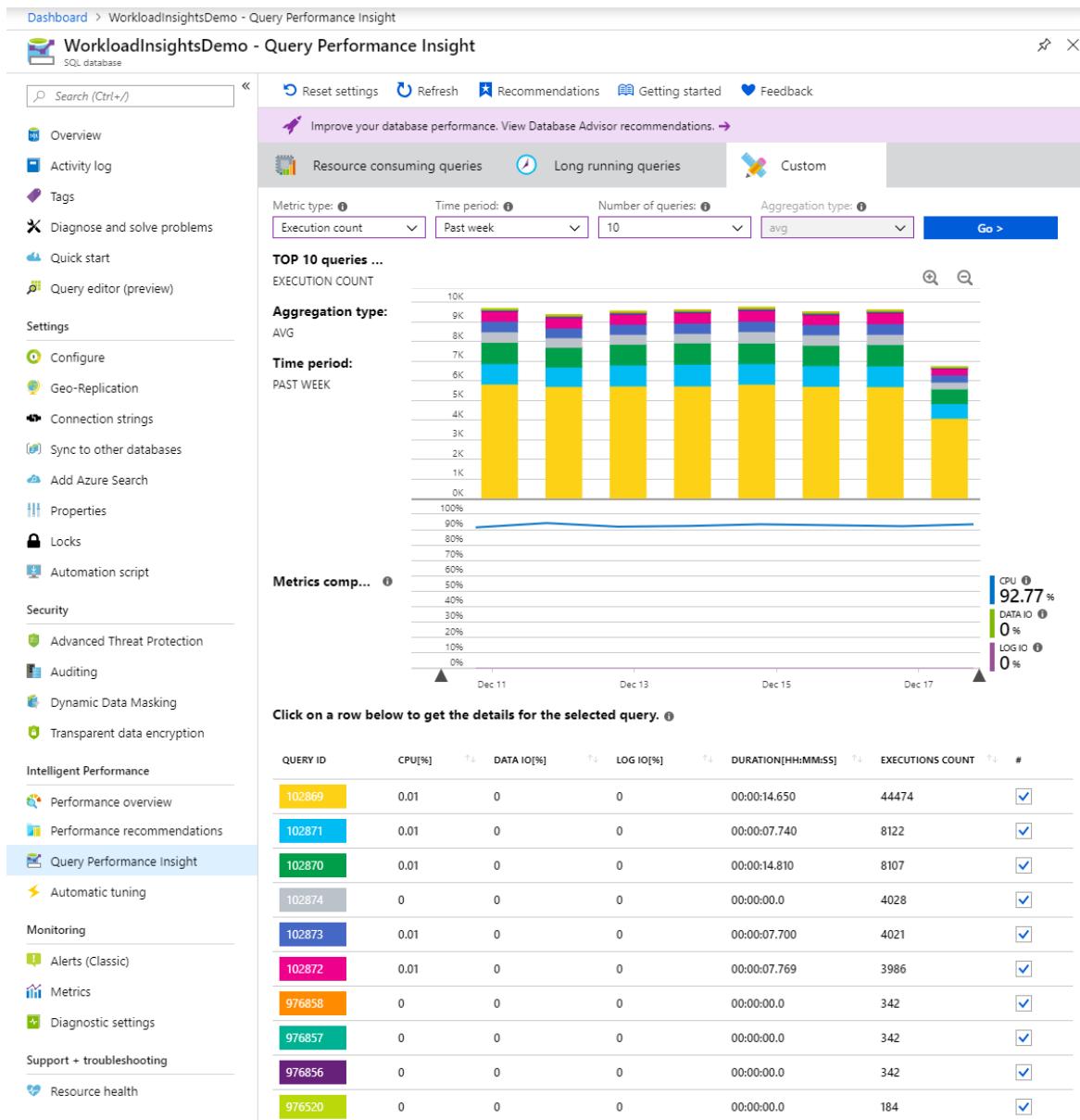
In some cases, a high execution count can lead to more network round trips. Round trips affect performance.

They're subject to network latency and to downstream server latency.

For example, many data-driven websites heavily access the database for every user request. Although connection pooling helps, the increased network traffic and processing load on the database server can slow performance. In general, keep round trips to a minimum.

To identify frequently executed ("chatty") queries:

1. Open the **Custom** tab in Query Performance Insight for the selected database.
2. Change the metrics to **execution count**.
3. Select the number of queries and the observation interval.
4. Select the **Go >** button to see the customized view.



## Understand performance tuning annotations

While exploring your workload in Query Performance Insight, you might notice icons with a vertical line on top of the chart.

These icons are annotations. They show performance recommendations from [SQL Database Advisor](#). By hovering over an annotation, you can get summarized information on performance recommendations.

Resource consuming queries

Custom

Metric type: CPU Time period: Past month Number of queries: 10 Aggregation type: sum Go >

8/8/2016 1:44:41 PM - Potential schema issue was detected. Click here for more information.

TOP 10 queries by CPU consumption

100%  
90%  
80%  
70%

If you want to understand more or apply the advisor's recommendation, select the icon to open details of the recommended action. If this is an active recommendation, you can apply it right away from the portal.

Feedback

Fix schema issues (preview)

Discard

RECOMMENDED ACTION: Fix schema issues (preview) Learn more

STATUS: Active

LAST UPDATE: 8/8/2016 1:28:01 PM

INITIATED BY: N/A

Estimated impact

IMPACT: HIGH

Number of errors

Category	Impact	Number of errors
Category 1	High	~10,000
Category 2	Medium	~8,000
Category 3	Low	~6,000
Category 4	Medium	~4,000
Category 5	High	~2,000
Category 6	Medium	~1,000
Category 7	Low	~8,000
Category 8	Medium	~6,000
Category 9	High	~4,000
Category 10	Medium	~2,000

In some cases, due to the zoom level, it's possible that annotations close to each other are collapsed into a single annotation. Query Performance Insight represents this as a group annotation icon. Selecting the group annotation icon opens a new blade that lists the annotations.

Correlating queries and performance-tuning actions might help you to better understand your workload.

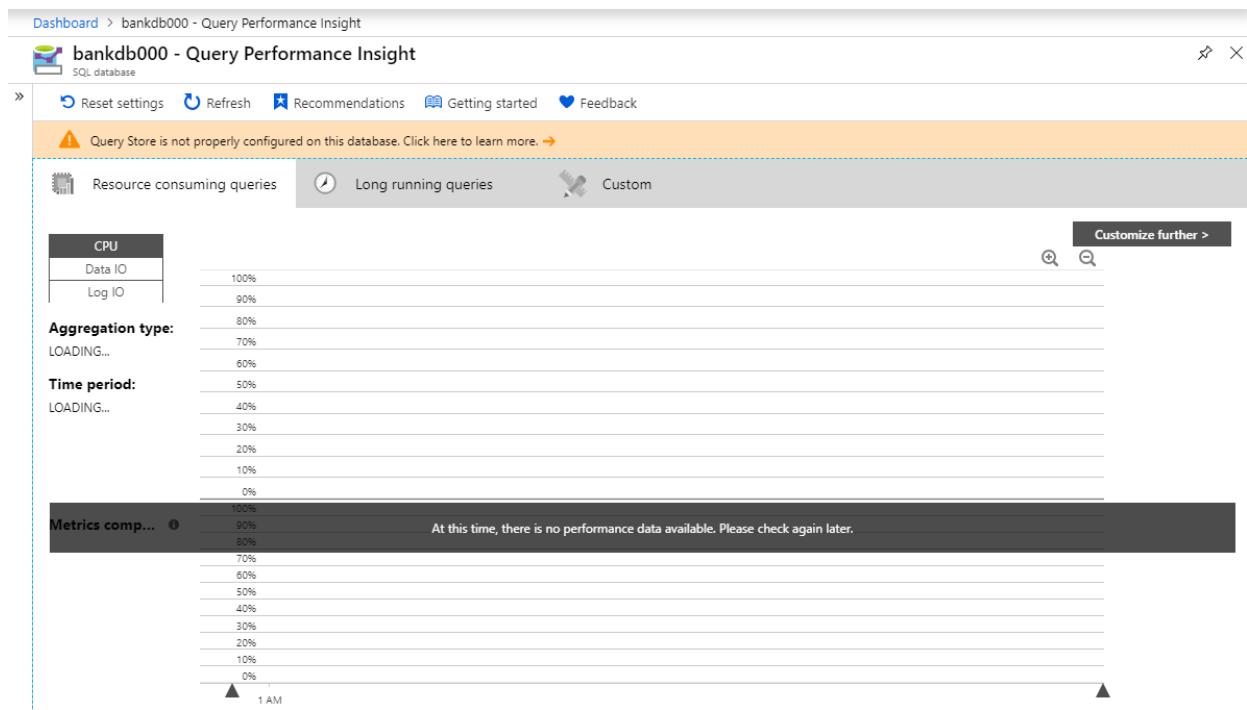
## Optimize the Query Store configuration for Query Performance Insight

While using Query Performance Insight, you might see the following Query Store error messages:

- "Query Store is not properly configured on this database. Click here to learn more."
- "Query Store is not properly configured on this database. Click here to change settings."

These messages usually appear when Query Store can't collect new data.

The first case happens when Query Store is in the read-only state and parameters are set optimally. You can fix this by increasing the size of the data store, or by clearing Query Store. (If you clear Query Store, all previously collected telemetry will be lost.)



The second case happens when Query Store is not enabled, or parameters are not set optimally. You can change the retention and capture policy, and also enable Query Store, by running the following commands provided from [SQL Server Management Studio \(SSMS\)](#) or the Azure portal.

### Recommended retention and capture policy

There are two types of retention policies:

- **Size based:** If this policy is set to **AUTO**, it will clean data automatically when near maximum size is reached.
- **Time based:** By default, this policy is set to 30 days. If Query Store runs out of space, it will delete query information older than 30 days.

You can set the capture policy to:

- **All:** Query Store captures all queries.
- **Auto:** Query Store ignores infrequent queries and queries with insignificant compile and execution duration. Thresholds for execution count, compile duration, and runtime duration are internally determined. This is the default option.
- **None:** Query Store stops capturing new queries, but runtime statistics for already captured queries are still collected.

We recommend setting all policies to **AUTO** and the cleaning policy to 30 days by executing the following commands from [SSMS](#) or the Azure portal. (Replace `[YourDB]` with the database name.)

```

ALTER DATABASE [YourDB]
SET QUERY_STORE (SIZE_BASED_CLEANUP_MODE = AUTO);

ALTER DATABASE [YourDB]
SET QUERY_STORE (CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30));

ALTER DATABASE [YourDB]
SET QUERY_STORE (QUERY_CAPTURE_MODE = AUTO);

```

Increase the size of Query Store by connecting to a database through [SSMS](#) or the Azure portal and running the following query. (Replace `[YourDB]` with the database name.)

```
ALTER DATABASE [YourDB]
SET QUERY_STORE (MAX_STORAGE_SIZE_MB = 1024);
```

Applying these settings will eventually make Query Store collect telemetry for new queries. If you need Query Store to be operational right away, you can optionally choose to clear Query Store by running the following query through SSMS or the Azure portal. (Replace `YourDB` with the database name.)

**NOTE**

Running the following query will delete all previously collected monitored telemetry in Query Store.

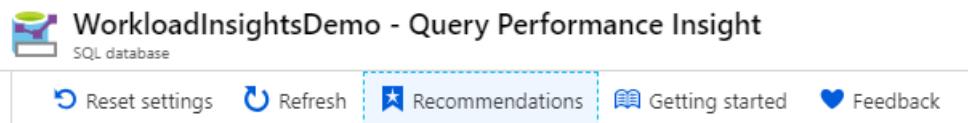
```
ALTER DATABASE [YourDB] SET QUERY_STORE CLEAR;
```

## Summary

Query Performance Insight helps you understand the impact of query workload and how it relates to the consumption of database resources. With this feature, you'll learn about the top-consuming queries on your database, and you'll find queries to optimize before they become a problem.

## Next steps

- For database performance recommendations, select [Recommendations](#) on the Query Performance Insight navigation blade.



- Consider enabling [Automatic Tuning](#) for common database performance problems.
- Learn how [Intelligent Insights](#) can help automatically troubleshoot database performance problems.
- Consider using [Azure SQL Analytics](#) for advanced performance monitoring of a large fleet of SQL databases, elastic pools, and Managed Instances with built-in intelligence.

# Resolving Transact-SQL differences during migration to SQL Database

11/7/2019 • 5 minutes to read • [Edit Online](#)

When migrating your database from SQL Server to Azure SQL Server, you may discover that your database requires some re-engineering before the SQL Server can be migrated. This article provides guidance to assist you in both performing this re-engineering and understanding the underlying reasons why the re-engineering is necessary. To detect incompatibilities, use the [Data Migration Assistant \(DMA\)](#).

## Overview

Most Transact-SQL features that applications use are fully supported in both Microsoft SQL Server and Azure SQL Database. For example, the core SQL components such as data types, operators, string, arithmetic, logical, and cursor functions, work identically in SQL Server and SQL Database. There are, however, a few T-SQL differences in DDL (data-definition language) and DML (data manipulation language) elements resulting in T-SQL statements and queries that are only partially supported (which we discuss later in this article).

In addition, there are some features and syntax that is not supported at all because Azure SQL Database is designed to isolate features from dependencies on the master database and the operating system. As such, most server-level activities are inappropriate for SQL Database. T-SQL statements and options are not available if they configure server-level options, operating system components, or specify file system configuration. When such capabilities are required, an appropriate alternative is often available in some other way from SQL Database or from another Azure feature or service.

For example, high availability is built into Azure SQL Database using technology similar to [Always On Availability Groups](#). T-SQL statements related to availability groups are not supported by SQL Database, and the dynamic management views related to Always On Availability Groups are also not supported.

For a list of the features that are supported and unsupported by SQL Database, see [Azure SQL Database feature comparison](#). The list on this page supplements that guidelines and features article, and focuses on Transact-SQL statements.

## Transact-SQL syntax statements with partial differences

The core DDL (data definition language) statements are available, but some DDL statements have extensions related to disk placement and unsupported features.

- CREATE and ALTER DATABASE statements have over three dozen options. The statements include file placement, FILESTREAM, and service broker options that only apply to SQL Server. This may not matter if you create databases before you migrate, but if you are migrating T-SQL code that creates databases you should compare [CREATE DATABASE \(Azure SQL Database\)](#) with the SQL Server syntax at [CREATE DATABASE \(SQL Server Transact-SQL\)](#) to make sure all the options you use are supported. CREATE DATABASE for Azure SQL Database also has service objective and elastic scale options that apply only to SQL Database.
- The CREATE and ALTER TABLE statements have FileTable options that cannot be used on SQL Database because FILESTREAM is not supported.
- CREATE and ALTER login statements are supported but SQL Database does not offer all the options. To make your database more portable, SQL Database encourages using contained database users instead of logins whenever possible. For more information, see [CREATE/ALTER LOGIN](#) and [Controlling and granting database](#)

access.

## Transact-SQL syntax not supported in Azure SQL Database

In addition to Transact-SQL statements related to the unsupported features described in [Azure SQL Database feature comparison](#), the following statements and groups of statements, are not supported. As such, if your database to be migrated is using any of the following features, re-engineer your T-SQL to eliminate these T-SQL features and statements.

- Collation of system objects
- Connection related: Endpoint statements. SQL Database does not support Windows authentication, but does support the similar Azure Active Directory authentication. Some authentication types require the latest version of SSMS. For more information, see [Connecting to SQL Database or SQL Data Warehouse By Using Azure Active Directory Authentication](#).
- Cross database queries using three or four part names. (Read-only cross-database queries are supported by using [elastic database query](#).)
- Cross database ownership chaining, `TRUSTWORTHY` setting
- `EXECUTE AS LOGIN` Use 'EXECUTE AS USER' instead.
- Encryption is supported except for extensible key management
- Eventing: Events, event notifications, query notifications
- File placement: Syntax related to database file placement, size, and database files that are automatically managed by Microsoft Azure.
- High availability: Syntax related to high availability, which is managed through your Microsoft Azure account. This includes syntax for backup, restore, Always On, database mirroring, log shipping, recovery modes.
- Log reader: Syntax that relies upon the log reader, which is not available on SQL Database: Push Replication, Change Data Capture. SQL Database can be a subscriber of a push replication article.
- Functions: `fn_get_sql` , `fn_virtualfilestats` , `fn_virtualservernodes`
- Hardware: Syntax related to hardware-related server settings: such as memory, worker threads, CPU affinity, trace flags. Use service tiers and compute sizes instead.
- `KILL STATS JOB`
- `OPENQUERY` , `OPENROWSET` , `OPENDATASOURCE` , and four-part names
- .NET Framework: CLR integration with SQL Server
- Semantic search
- Server credentials: Use [database scoped credentials](#) instead.
- Server-level items: Server roles, `sys.login_token` . `GRANT` , `REVOKE` , and `DENY` of server level permissions are not available though some are replaced by database-level permissions. Some useful server-level DMVs have equivalent database-level DMVs.
- `SET REMOTE_PROC_TRANSACTIONS`
- `SHUTDOWN`
- `sp_addmessage`
- `sp_configure` options and `RECONFIGURE` . Some options are available using [ALTER DATABASE SCOPED CONFIGURATION](#).
- `sp_helpuser`
- `sp_migrate_user_to_contained`
- SQL Server Agent: Syntax that relies upon the SQL Server Agent or the MSDB database: alerts, operators, central management servers. Use scripting, such as Azure PowerShell instead.
- SQL Server audit: Use SQL Database auditing instead.
- SQL Server trace
- Trace flags: Some trace flag items have been moved to compatibility modes.

- Transact-SQL debugging
- Triggers: Server-scoped or logon triggers
- `USE` statement: To change the database context to a different database, you must make a new connection to the new database.

## Full Transact-SQL reference

For more information about Transact-SQL grammar, usage, and examples, see [Transact-SQL Reference \(Database Engine\)](#) in SQL Server Books Online.

### About the "Applies to" tags

The Transact-SQL reference includes articles related to SQL Server versions 2008 to the present. Below the article title there is an icon bar, listing the four SQL Server platforms, and indicating applicability. For example, availability groups were introduced in SQL Server 2012. The [CREATE AVAILABILITY GROUP](#) article indicates that the statement applies to **SQL Server (starting with 2012)**. The statement does not apply to SQL Server 2008, SQL Server 2008 R2, Azure SQL Database, Azure SQL Data Warehouse, or Parallel Data Warehouse.

In some cases, the general subject of an article can be used in a product, but there are minor differences between products. The differences are indicated at midpoints in the article as appropriate. In some cases, the general subject of an article can be used in a product, but there are minor differences between products. The differences are indicated at midpoints in the article as appropriate. For example the CREATE TRIGGER article is available in SQL Database. But the **ALL SERVER** option for server-level triggers, indicates that server-level triggers cannot be used in SQL Database. Use database-level triggers instead.

## Next steps

For a list of the features that are supported and unsupported by SQL Database, see [Azure SQL Database feature comparison](#). The list on this page supplements that guidelines and features article, and focuses on Transact-SQL statements.

# Sync data across multiple cloud and on-premises databases with SQL Data Sync

11/22/2019 • 10 minutes to read • [Edit Online](#)

SQL Data Sync is a service built on Azure SQL Database that lets you synchronize the data you select bi-directionally across multiple SQL databases and SQL Server instances.

## IMPORTANT

Azure SQL Data Sync does not support Azure SQL Database Managed Instance at this time.

## When to use Data Sync

Data Sync is useful in cases where data needs to be kept updated across several Azure SQL databases or SQL Server databases. Here are the main use cases for Data Sync:

- **Hybrid Data Synchronization:** With Data Sync, you can keep data synchronized between your on-premises databases and Azure SQL databases to enable hybrid applications. This capability may appeal to customers who are considering moving to the cloud and would like to put some of their application in Azure.
- **Distributed Applications:** In many cases, it's beneficial to separate different workloads across different databases. For example, if you have a large production database, but you also need to run a reporting or analytics workload on this data, it's helpful to have a second database for this additional workload. This approach minimizes the performance impact on your production workload. You can use Data Sync to keep these two databases synchronized.
- **Globally Distributed Applications:** Many businesses span several regions and even several countries/regions. To minimize network latency, it's best to have your data in a region close to you. With Data Sync, you can easily keep databases in regions around the world synchronized.

Data Sync isn't the preferred solution for the following scenarios:

SCENARIO	SOME RECOMMENDED SOLUTIONS
Disaster Recovery	<a href="#">Azure geo-redundant backups</a>
Read Scale	<a href="#">Use read-only replicas to load balance read-only query workloads (preview)</a>
ETL (OLTP to OLAP)	<a href="#">Azure Data Factory</a> or <a href="#">SQL Server Integration Services</a>
Migration from on-premises SQL Server to Azure SQL Database	<a href="#">Azure Database Migration Service</a>

## Overview of SQL Data Sync

Data Sync is based around the concept of a Sync Group. A Sync Group is a group of databases that you want to synchronize.

Data Sync uses a hub and spoke topology to synchronize data. You define one of the databases in the sync

group as the Hub Database. The rest of the databases are member databases. Sync occurs only between the Hub and individual members.

- The **Hub Database** must be an Azure SQL Database.
- The **member databases** can be either SQL Databases, on-premises SQL Server databases, or SQL Server instances on Azure virtual machines.
- The **Sync Database** contains the metadata and log for Data Sync. The Sync Database has to be an Azure SQL Database located in the same region as the Hub Database. The Sync Database is customer created and customer owned.

#### NOTE

If you're using an on premises database as a member database, you have to [install and configure a local sync agent](#).

Figure 1. Sync Between 2 Databases

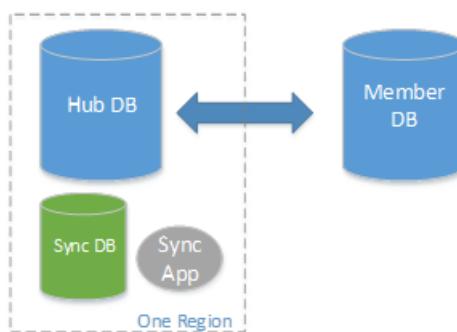
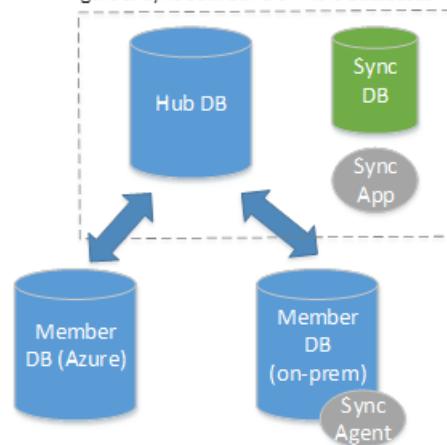


Figure 2. Sync Between 3 or more Databases



A Sync Group has the following properties:

- The **Sync Schema** describes which data is being synchronized.
- The **Sync Direction** can be bi-directional or can flow in only one direction. That is, the Sync Direction can be *Hub to Member*, or *Member to Hub*, or both.
- The **Sync Interval** describes how often synchronization occurs.
- The **Conflict Resolution Policy** is a group level policy, which can be *Hub wins* or *Member wins*.

## How does Data Sync work

- **Tracking data changes:** Data Sync tracks changes using insert, update, and delete triggers. The changes are recorded in a side table in the user database. Note that BULK INSERT doesn't fire triggers by default. If FIRE\_TRIGGERS isn't specified, no insert triggers execute. Add the FIRE\_TRIGGERS option so Data Sync can track those inserts.
- **Synchronizing data:** Data Sync is designed in a Hub and Spoke model. The Hub syncs with each member individually. Changes from the Hub are downloaded to the member and then changes from the member are uploaded to the Hub.
- **Resolving conflicts:** Data Sync provides two options for conflict resolution, *Hub wins* or *Member wins*.
  - If you select *Hub wins*, the changes in the hub always overwrite changes in the member.
  - If you select *Member wins*, the changes in the member overwrite changes in the hub. If there's more than one member, the final value depends on which member syncs first.

## Compare Data Sync with Transactional Replication

	DATA SYNC	TRANSACTIONAL REPLICATION
Advantages	<ul style="list-style-type: none"> <li>- Active-active support</li> <li>- Bi-directional between on-premises and Azure SQL Database</li> </ul>	<ul style="list-style-type: none"> <li>- Lower latency</li> <li>- Transactional consistency</li> <li>- Reuse existing topology after migration</li> </ul>
Disadvantages	<ul style="list-style-type: none"> <li>- 5 min or more latency</li> <li>- No transactional consistency</li> <li>- Higher performance impact</li> </ul>	<ul style="list-style-type: none"> <li>- Can't publish from Azure SQL Database single database or pooled database</li> <li>- High maintenance cost</li> </ul>

## Get started with SQL Data Sync

### Set up Data Sync in the Azure portal

- [Set up Azure SQL Data Sync](#)
- Data Sync Agent - [Data Sync Agent for Azure SQL Data Sync](#)

### Set up Data Sync with PowerShell

- [Use PowerShell to sync between multiple Azure SQL databases](#)
- [Use PowerShell to sync between an Azure SQL Database and a SQL Server on-premises database](#)

### Review the best practices for Data Sync

- [Best practices for Azure SQL Data Sync](#)

### Did something go wrong

- [Troubleshoot issues with Azure SQL Data Sync](#)

## Consistency and performance

### Eventual consistency

Since Data Sync is trigger-based, transactional consistency isn't guaranteed. Microsoft guarantees that all changes are made eventually and that Data Sync doesn't cause data loss.

### Performance impact

Data Sync uses insert, update, and delete triggers to track changes. It creates side tables in the user database for change tracking. These change tracking activities have an impact on your database workload. Assess your service tier and upgrade if needed.

Provisioning and deprovisioning during sync group creation, update, and deletion may also impact the database performance.

## Requirements and limitations

### General requirements

- Each table must have a primary key. Don't change the value of the primary key in any row. If you have to change a primary key value, delete the row and recreate it with the new primary key value.

## **IMPORTANT**

Changing the value of an existing primary key will result in the following faulty behavior:

- Data between hub and member can be lost even though sync does not report any issue.
- Sync can fail because the tracking table has a non-existing row from source due to the primary key change.

- Snapshot isolation must be enabled. For more info, see [Snapshot Isolation in SQL Server](#).

## **General limitations**

- A table can't have an identity column that isn't the primary key.
- A primary key can't have the following data types: sql\_variant, binary, varbinary, image, xml.
- Be cautious when you use the following data types as a primary key, because the supported precision is only to the second: time, datetime, datetime2, datetimeoffset.
- The names of objects (databases, tables, and columns) can't contain the printable characters period (.), left square bracket ([), or right square bracket (]).
- Azure Active Directory authentication isn't supported.
- Tables with same name but different schema (for example, dbo.customers and sales.customers) aren't supported.
- Columns with User Defined Data Types aren't supported

## **Unsupported data types**

- FileStream
- SQL/CLR UDT
- XMLSchemaCollection (XML supported)
- Cursor, RowVersion, Timestamp, Hierarchyid

## **Unsupported column types**

Data Sync can't sync read-only or system-generated columns. For example:

- Computed columns.
- System-generated columns for temporal tables.

## **Limitations on service and database dimensions**

DIMENSIONS	LIMIT	WORKAROUND
Maximum number of sync groups any database can belong to.	5	
Maximum number of endpoints in a single sync group	30	
Maximum number of on-premises endpoints in a single sync group.	5	Create multiple sync groups
Database, table, schema, and column names	50 characters per name	
Tables in a sync group	500	Create multiple sync groups
Columns in a table in a sync group	1000	
Data row size on a table	24 Mb	

DIMENSIONS	LIMIT	WORKAROUND
Minimum sync interval	5 Minutes	

#### NOTE

There may be up to 30 endpoints in a single sync group if there is only one sync group. If there is more than one sync group, the total number of endpoints across all sync groups cannot exceed 30. If a database belongs to multiple sync groups, it is counted as multiple endpoints, not one.

## FAQ about SQL Data Sync

### How much does the SQL Data Sync service cost

There's no charge for the SQL Data Sync service itself. However, you still collect data transfer charges for data movement in and out of your SQL Database instance. For more info, see [SQL Database pricing](#).

### What regions support Data Sync

SQL Data Sync is available in all regions.

### Is a SQL Database account required

Yes. You must have a SQL Database account to host the Hub Database.

### Can I use Data Sync to sync between SQL Server on-premises databases only

Not directly. You can sync between SQL Server on-premises databases indirectly, however, by creating a Hub database in Azure, and then adding the on-premises databases to the sync group.

### Can I use Data Sync to sync between SQL Databases that belong to different subscriptions

Yes. You can sync between SQL Databases that belong to resource groups owned by different subscriptions.

- If the subscriptions belong to the same tenant, and you have permission to all subscriptions, you can configure the sync group in the Azure portal.
- Otherwise, you have to use PowerShell to add the sync members that belong to different subscriptions.

### Can I use Data Sync to sync between SQL Databases that belong to different clouds (like Azure Public Cloud and Azure China 21Vianet)

Yes. You can sync between SQL Databases that belong to different clouds, you have to use PowerShell to add the sync members that belong to the different subscriptions.

### Can I use Data Sync to seed data from my production database to an empty database, and then sync them

Yes. Create the schema manually in the new database by scripting it from the original. After you create the schema, add the tables to a sync group to copy the data and keep it synced.

### Should I use SQL Data Sync to back up and restore my databases

It isn't recommended to use SQL Data Sync to create a backup of your data. You can't back up and restore to a specific point in time because SQL Data Sync synchronizations are not versioned. Furthermore, SQL Data Sync does not back up other SQL objects, such as stored procedures, and doesn't do the equivalent of a restore operation quickly.

For one recommended backup technique, see [Copy an Azure SQL database](#).

### Can Data Sync sync encrypted tables and columns

- If a database uses Always Encrypted, you can sync only the tables and columns that are *not* encrypted. You can't sync the encrypted columns, because Data Sync can't decrypt the data.
- If a column uses Column-Level Encryption (CLE), you can sync the column, as long as the row size is less

than the maximum size of 24 Mb. Data Sync treats the column encrypted by key (CLE) as normal binary data. To decrypt the data on other sync members, you need to have the same certificate.

### **Is collation supported in SQL Data Sync**

Yes. SQL Data Sync supports collation in the following scenarios:

- If the selected sync schema tables aren't already in your hub or member databases, then when you deploy the sync group, the service automatically creates the corresponding tables and columns with the collation settings selected in the empty destination databases.
- If the tables to be synced already exist in both your hub and member databases, SQL Data Sync requires that the primary key columns have the same collation between hub and member databases to successfully deploy the sync group. There are no collation restrictions on columns other than the primary key columns.

### **Is federation supported in SQL Data Sync**

Federation Root Database can be used in the SQL Data Sync Service without any limitation. You can't add the Federated Database endpoint to the current version of SQL Data Sync.

## Next steps

### **Update the schema of a synced database**

Do you have to update the schema of a database in a sync group? Schema changes aren't automatically replicated. For some solutions, see the following articles:

- [Automate the replication of schema changes in Azure SQL Data Sync](#)
- [Use PowerShell to update the sync schema in an existing sync group](#)

### **Monitor and troubleshoot**

Is SQL Data Sync doing as expected? To monitor activity and troubleshoot issues, see the following articles:

- [Monitor Azure SQL Data Sync with Azure Monitor logs](#)
- [Troubleshoot issues with Azure SQL Data Sync](#)

### **Learn more about Azure SQL Database**

For more info about SQL Database, see the following articles:

- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Managing Azure SQL databases using Azure Automation

11/7/2019 • 2 minutes to read • [Edit Online](#)

This guide will introduce you to the Azure Automation service, and how it can be used to simplify management of your Azure SQL databases.

## What is Azure Automation?

[Azure Automation](#) is an Azure service for simplifying cloud management through process automation. Using Azure Automation, long-running, manual, error-prone, and frequently repeated tasks can be automated to increase reliability, efficiency, and time to value for your organization.

Azure Automation provides a workflow execution engine with high reliability and high availability, and that scales to meet your needs as your organization grows. In Azure Automation, processes can be kicked off manually, by third-party systems, or at scheduled intervals so that tasks happen exactly when needed.

Lower operational overhead and free up IT / DevOps staff to focus on work that adds business value by moving your cloud management tasks to be run automatically by Azure Automation.

## How can Azure Automation help manage Azure SQL databases?

Azure SQL Database can be managed in Azure Automation by using the [Azure SQL Database PowerShell cmdlets](#) that are available in the [Azure PowerShell tools](#). Azure Automation has these Azure SQL Database PowerShell cmdlets available out of the box, so that you can perform all of your SQL DB management tasks within the service. You can also pair these cmdlets in Azure Automation with the cmdlets for other Azure services, to automate complex tasks across Azure services and across third-party systems.

Azure Automation also has the ability to communicate with SQL servers directly, by issuing SQL commands using PowerShell.

The [Azure Automation runbook gallery](#) contains a variety of product team and community runbooks to get started automating management of Azure SQL databases, other Azure services, and third-party systems. Gallery runbooks include:

- [Run SQL queries against a SQL Server database](#)
- [Vertically scale \(up or down\) an Azure SQL Database on a schedule](#)
- [Truncate a SQL table if its database approaches its maximum size](#)
- [Index tables in an Azure SQL Database if they are highly fragmented](#)

## Next steps

Now that you've learned the basics of Azure Automation and how it can be used to manage Azure SQL databases, follow these links to learn more about Azure Automation.

- [Azure Automation Overview](#)
- [My first runbook](#)
- [Azure Automation: Your SQL Agent in the Cloud](#)

# Create and manage SQL Database servers and single databases in Azure SQL Database

11/7/2019 • 8 minutes to read • [Edit Online](#)

You can create and manage SQL Database servers and single databases using the Azure portal, PowerShell, Azure CLI, REST API, and Transact-SQL.

## Azure portal: Manage SQL Database servers and single databases

You can create the Azure SQL database's resource group ahead of time or while creating the server itself. There are multiple methods for getting to a new SQL server form, either by creating a new SQL server or as part of creating a new database.

### Create a blank SQL Database server

To create a SQL Database server using the [Azure portal](#), navigate to a blank SQL server (logical server) form.

### Create a blank or sample SQL single database

To create an Azure SQL single database using the [Azure portal](#), navigate to a blank SQL Database form and provide the requested information. You can create the Azure SQL database's resource group and SQL Database server ahead of time or while creating the single database itself. You can create a blank database or create a sample database based on Adventure Works LT.

The screenshot shows the Azure portal's 'New' blade. On the left is a navigation sidebar with a 'Create a resource' button highlighted with a red box. The main area has a search bar labeled 'Search the Marketplace'. Below it are two tabs: 'Azure Marketplace' and 'Featured', with 'Featured' selected. A red box highlights the 'SQL Database' item under the 'Featured' tab, which includes a 'Quickstart tutorial' link. Another red box highlights the 'Databases' category in the list of service types. The list also includes 'Compute', 'Networking', 'Storage', 'Web', 'Mobile', 'Containers', 'Analytics', 'AI + Machine Learning', 'Internet of Things', 'Integration', 'Security', and 'Identity'. Each item has a small icon and a 'Quickstart tutorial' link.

Service Type	Description	Quickstart Tutorial
Azure SQL Managed Instance	Azure SQL Managed Instance	<a href="#">Quickstart tutorial</a>
SQL Database	SQL Database	<a href="#">Quickstart tutorial</a>
SQL Data Warehouse	SQL Data Warehouse	<a href="#">Quickstart tutorial</a>
SQL Elastic database pool	SQL Elastic database pool	<a href="#">Learn more</a>
Azure Database for MariaDB	Azure Database for MariaDB	<a href="#">Learn more</a>
Azure Database for MySQL	Azure Database for MySQL	<a href="#">Quickstart tutorial</a>
Azure Database for PostgreSQL	Azure Database for PostgreSQL	<a href="#">Quickstart tutorial</a>

## IMPORTANT

For information on selecting the pricing tier for your database, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#).

To create a Managed Instance, see [Create a Managed Instance](#)

## Manage an existing SQL Database server

To manage an existing SQL Database server, navigate to the server using a number of methods - such as from specific SQL database page, the **SQL servers** page, or the **All resources** page.

To manage an existing database, navigate to the **SQL databases** page and click the database you wish to manage. The following screenshot shows how to begin setting a server-level firewall for a database from the **Overview** page for a database.

The screenshot shows the Azure portal interface for managing a SQL database. On the left, there's a sidebar with various icons and a resource group named 'myResourceGroup'. The main area shows the 'Firewall settings' for a database named 'mynewserver-20181208'. A table lists a single rule: 'ClientIP/Address\_2018-11' with 'Start IP' and 'End IP' both set to '73.42.249.7'. There are buttons for 'Save' and 'Discard', with 'Save' being highlighted by a red box. Other buttons like 'Copy', 'Restore', 'Export', and 'Connect with...' are also visible.

## IMPORTANT

To configure performance properties for a database, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#).

## TIP

For an Azure portal quickstart, see [Create an Azure SQL database in the Azure portal](#).

## PowerShell: Manage SQL Database servers and single databases

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

**IMPORTANT**

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

To create and manage Azure SQL Database servers, single and pooled databases, and SQL Database server firewalls with Azure PowerShell, use the following PowerShell cmdlets. If you need to install or upgrade PowerShell, see [Install Azure PowerShell module](#).

**TIP**

For PowerShell example scripts, see [Use PowerShell to create an Azure SQL single database and configure a SQL Database server firewall rule](#) and [Monitor and scale a SQL single database using PowerShell](#).

CMDLET	DESCRIPTION
<a href="#">New-AzSqlDatabase</a>	Creates a database
<a href="#">Get-AzSqlDatabase</a>	Gets one or more databases
<a href="#">Set-AzSqlDatabase</a>	Sets properties for a database, or moves an existing database into an elastic pool
<a href="#">Remove-AzSqlDatabase</a>	Removes a database
<a href="#">New-AzResourceGroup</a>	Creates a resource group
<a href="#">New-AzSqlServer</a>	Creates a server
<a href="#">Get-AzSqlServer</a>	Returns information about servers
<a href="#">Set-AzSqlServer</a>	Modifies properties of a server
<a href="#">Remove-AzSqlServer</a>	Removes a server
<a href="#">New-AzSqlServerFirewallRule</a>	Creates a server-level firewall rule
<a href="#">Get-AzSqlServerFirewallRule</a>	Gets firewall rules for a server
<a href="#">Set-AzSqlServerFirewallRule</a>	Modifies a firewall rule in a server
<a href="#">Remove-AzSqlServerFirewallRule</a>	Deletes a firewall rule from a server.
<a href="#">New-AzSqlServerVirtualNetworkRule</a>	Creates a <i>virtual network rule</i> , based on a subnet that is a Virtual Network service endpoint.

## Azure CLI: Manage SQL Database servers and single databases

To create and manage Azure SQL server, databases, and firewalls with [Azure CLI](#), use the following [Azure CLI SQL Database](#) commands. Use the [Cloud Shell](#) to run the CLI in your browser, or [install](#) it on macOS, Linux, or Windows. For creating and managing elastic pools, see [Elastic pools](#).

**TIP**

For an Azure CLI quickstart, see [Create an Azure SQL single database using the Azure CLI](#). For Azure CLI example scripts, see [Use CLI to create an Azure SQL single database and configure a SQL Database firewall rule](#) and [Use CLI to monitor and scale an Azure SQL single database](#).

CMDLET	DESCRIPTION
<a href="#">az sql db create</a>	Creates a database
<a href="#">az sql db list</a>	Lists all databases and data warehouses in a server, or all databases in an elastic pool
<a href="#">az sql db list-editions</a>	Lists available service objectives and storage limits
<a href="#">az sql db list-usages</a>	Returns database usages
<a href="#">az sql db show</a>	Gets a database or data warehouse
<a href="#">az sql db update</a>	Updates a database
<a href="#">az sql db delete</a>	Removes a database
<a href="#">az group create</a>	Creates a resource group
<a href="#">az sql server create</a>	Creates a server
<a href="#">az sql server list</a>	Lists servers
<a href="#">az sql server list-usages</a>	Returns server usages
<a href="#">az sql server show</a>	Gets a server
<a href="#">az sql server update</a>	Updates a server
<a href="#">az sql server delete</a>	Deletes a server
<a href="#">az sql server firewall-rule create</a>	Creates a server firewall rule
<a href="#">az sql server firewall-rule list</a>	Lists the firewall rules on a server
<a href="#">az sql server firewall-rule show</a>	Shows the detail of a firewall rule
<a href="#">az sql server firewall-rule update</a>	Updates a firewall rule
<a href="#">az sql server firewall-rule delete</a>	Deletes a firewall rule

## Transact-SQL: Manage SQL Database servers and single databases

To create and manage Azure SQL server, databases, and firewalls with Transact-SQL, use the following T-SQL commands. You can issue these commands using the Azure portal, [SQL Server Management Studio](#), [Visual Studio Code](#), or any other program that can connect to an Azure SQL Database server and pass Transact-SQL

commands. For managing elastic pools, see [Elastic pools](#).

**TIP**

For a quickstart using SQL Server Management Studio on Microsoft Windows, see [Azure SQL Database: Use SQL Server Management Studio to connect and query data](#). For a quickstart using Visual Studio Code on the macOS, Linux, or Windows, see [Azure SQL Database: Use Visual Studio Code to connect and query data](#).

**IMPORTANT**

You cannot create or delete a server using Transact-SQL.

COMMAND	DESCRIPTION
<a href="#">CREATE DATABASE</a>	Creates a new single database. You must be connected to the master database to create a new database.
<a href="#">ALTER DATABASE (Azure SQL Database)</a>	Modifies an Azure SQL database.
<a href="#">DROP DATABASE (Transact-SQL)</a>	Deletes a database.
<a href="#">sys.database_service_objectives (Azure SQL Database)</a>	Returns the edition (service tier), service objective (pricing tier), and elastic pool name, if any, for an Azure SQL database or an Azure SQL Data Warehouse. If logged on to the master database in an Azure SQL Database server, returns information on all databases. For Azure SQL Data Warehouse, you must be connected to the master database.
<a href="#">sys.dm_db_resource_stats (Azure SQL Database)</a>	Returns CPU, IO, and memory consumption for an Azure SQL Database database. One row exists for every 15 seconds, even if there is no activity in the database.
<a href="#">sys.resource_stats (Azure SQL Database)</a>	Returns CPU usage and storage data for an Azure SQL Database. The data is collected and aggregated within five-minute intervals.
<a href="#">sys.database_connection_stats (Azure SQL Database)</a>	Contains statistics for SQL Database database connectivity events, providing an overview of database connection successes and failures.
<a href="#">sys.event_log (Azure SQL Database)</a>	Returns successful Azure SQL Database database connections, connection failures, and deadlocks. You can use this information to track or troubleshoot your database activity with SQL Database.
<a href="#">sp_set_firewall_rule (Azure SQL Database)</a>	Creates or updates the server-level firewall settings for your SQL Database server. This stored procedure is only available in the master database to the server-level principal login. A server-level firewall rule can only be created using Transact-SQL after the first server-level firewall rule has been created by a user with Azure-level permissions
<a href="#">sys.firewall_rules (Azure SQL Database)</a>	Returns information about the server-level firewall settings associated with your Microsoft Azure SQL Database.

COMMAND	DESCRIPTION
<a href="#">sp_delete_firewall_rule (Azure SQL Database)</a>	Removes server-level firewall settings from your SQL Database server. This stored procedure is only available in the master database to the server-level principal login.
<a href="#">sp_set_database_firewall_rule (Azure SQL Database)</a>	Creates or updates the database-level firewall rules for your Azure SQL Database or SQL Data Warehouse. Database firewall rules can be configured for the master database, and for user databases on SQL Database. Database firewall rules are useful when using contained database users.
<a href="#">sys.database_firewall_rules (Azure SQL Database)</a>	Returns information about the database-level firewall settings associated with your Microsoft Azure SQL Database.
<a href="#">sp_delete_database_firewall_rule (Azure SQL Database)</a>	Removes database-level firewall setting from your Azure SQL Database or SQL Data Warehouse.

## REST API: Manage SQL Database servers and single databases

To create and manage Azure SQL server, databases, and firewalls, use these REST API requests.

COMMAND	DESCRIPTION
<a href="#">Servers - Create or update</a>	Creates or updates a new server.
<a href="#">Servers - Delete</a>	Deletes a SQL server.
<a href="#">Servers - Get</a>	Gets a server.
<a href="#">Servers - List</a>	Returns a list of servers in a subscription.
<a href="#">Servers - List by resource group</a>	Returns a list of servers in a resource group.
<a href="#">Servers - Update</a>	Updates an existing server.
<a href="#">Databases - Create or update</a>	Creates a new database or updates an existing database.
<a href="#">Databases - Delete</a>	Deletes a database.
<a href="#">Databases - Get</a>	Gets a database.
<a href="#">Databases - List by elastic pool</a>	Returns a list of databases in an elastic pool.
<a href="#">Databases - List by server</a>	Returns a list of databases in a server.
<a href="#">Databases - Update</a>	Updates an existing database.
<a href="#">Firewall rules - Create or update</a>	Creates or updates a firewall rule.
<a href="#">Firewall rules - Delete</a>	Deletes a firewall rule.
<a href="#">Firewall rules - Get</a>	Gets a firewall rule.

COMMAND	DESCRIPTION
<a href="#">Firewall rules - List by server</a>	Returns a list of firewall rules.

## Next steps

- To learn about migrating a SQL Server database to Azure, see [Migrate to Azure SQL Database](#).
- For information about supported features, see [Features](#).

# Azure SQL Database Machine Learning Services with R (preview)

11/25/2019 • 2 minutes to read • [Edit Online](#)

Machine Learning Services is a feature of Azure SQL Database, used for executing in-database R scripts. The feature includes Microsoft R packages for high-performance predictive analytics and machine learning. The relational data can be used in R scripts through stored procedures, T-SQL script containing R statements, or R code containing T-SQL.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## NOTE

The preview is available for single databases and elastic pools using the vCore-based purchasing model in the **general purpose** and **business critical** service tiers. In this initial preview, the **hyperscale** service tier and the **managed instance** deployment option are not supported. Currently, R is the only supported language. There is no support for Python at this time.

The preview is currently available in the following regions: West Europe, North Europe, West US 2, East US, South Central US, North Central US, Canada Central, Southeast Asia, India South, and Australia Southeast.

## What you can do with R

Use the power of R language to deliver advanced analytics and machine learning in-database. This ability brings calculations and processing to where the data resides, eliminating the need to pull data across the network. Also, you can leverage the power of enterprise R packages to deliver advanced analytics at scale.

Machine Learning Services includes a base distribution of R, overlaid with enterprise R packages from Microsoft. Microsoft's R functions and algorithms are engineered for both scale and utility, delivering predictive analytics, statistical modeling, data visualizations, and leading-edge machine learning algorithms.

### R packages

Most common open-source R packages are pre-installed in Machine Learning Services. The following R packages from Microsoft are also included:

R PACKAGE	DESCRIPTION
<a href="#">Microsoft R Open</a>	Microsoft R Open is the enhanced distribution of R from Microsoft. It is a complete open-source platform for statistical analysis and data science. It is based on and 100% compatible with R, and includes additional capabilities for improved performance and reproducibility.

R PACKAGE	DESCRIPTION
RevoScaleR	RevoScaleR is the primary library for scalable R. Functions in this library are among the most widely used. Data transformations and manipulation, statistical summarization, visualization, and many forms of modeling and analyses are found in these libraries. Additionally, functions in these libraries automatically distribute workloads across available cores for parallel processing, with the ability to work on chunks of data that are coordinated and managed by the calculation engine.
MicrosoftML (R)	MicrosoftML adds machine learning algorithms to create custom models for text analysis, image analysis, and sentiment analysis.

In addition to the pre-installed packages, you can [install additional packages](#).

## Sign up for the preview

### IMPORTANT

Sign up for Azure SQL Database Machine Learning Services (preview) is currently closed.

Machine Learning Services with R is not recommended for production workload during the preview.

## Next steps

- See the [key differences from SQL Server Machine Learning Services](#).
- To learn how to use R to query Azure SQL Database Machine Learning Services (preview), see the [Quickstart guide](#).
- To get started with some simple R scripts, see [Create and run simple R scripts in Azure SQL Database Machine Learning Services \(preview\)](#).

# Key differences between Machine Learning Services in Azure SQL Database (preview) and SQL Server

11/25/2019 • 2 minutes to read • [Edit Online](#)

The functionality of Azure SQL Database Machine Learning Services (with R) in (preview) is similar to [SQL Server Machine Learning Services](#). Below are some key differences.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Language support

SQL Server has support for R and Python through the [extensibility framework](#). SQL Database does not support both languages. The key differences are:

- R is the only supported language in SQL Database. There is no support for Python at this time.
- The R version is 3.4.4.
- There is no need to configure `external scripts enabled` via `sp_configure`. Once you are [signed up](#), machine learning is enabled for your SQL database.

## Package management

R package management and installation work different between SQL Database and SQL Server. These differences are:

- R packages are installed via [sqlmlutils](#) or [CREATE EXTERNAL LIBRARY](#).
- Packages cannot perform outbound network calls. This limitation is similar to the [default firewall rules for Machine Learning Services](#) in SQL Server, but can't be changed in SQL Database.
- There is no support for packages that depend on external runtimes (like Java) or need access to OS APIs for installation or usage.

## Writing to a temporary table

If you're using RODBC in Azure SQL Database, then you can't write to a temporary table, whether it's created inside or outside of the `sp_execute_external_script` session. The workaround is to use `RxDbcData` and `rxDataStep` (with `overwrite=False` and `append="rows"`) to write to a global temporary table created before the `sp_execute_external_script` query.

## Resource governance

It is not possible to limit R resources through [Resource Governor](#) and external resource pools.

During the public preview, R resources are set to a maximum of 20% of the SQL Database resources, and depend on which service tier you choose. For more information, see [Azure SQL Database purchasing models](#).

## Insufficient memory error

If there is insufficient memory available for R, you will get an error message. Common error messages are:

- Unable to communicate with the runtime for 'R' script for request id: \*\*\*\*\*. Please check the requirements of 'R' runtime
- 'R' script error occurred during execution of 'sp\_execute\_external\_script' with HRESULT 0x80004004. ...an external script error occurred: "..could not allocate memory (0 Mb) in C function 'R\_AllocStringBuffer'"
- An external script error occurred: Error: cannot allocate vector of size.

Memory usage depends on how much is used in your R scripts and the number of parallel queries being executed.

If you receive the errors above, you can scale your database to a higher service tier to resolve this.

## Next steps

- See the overview, [Azure SQL Database Machine Learning Services with R \(preview\)](#).
- To learn how to use R to query Azure SQL Database Machine Learning Services (preview), see the [Quickstart guide](#).
- To get started with some simple R scripts, see [Create and run simple R scripts in Azure SQL Database Machine Learning Services \(preview\)](#).

# Quickstart: Create and run simple R scripts in Azure SQL Database Machine Learning Services (preview)

1/26/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you create and run a set of R scripts using Machine Learning Services (with R) in Azure SQL Database.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An [Azure SQL database](#) with a [server-level firewall rule](#)
- [Machine Learning Services](#) with R enabled. [Sign up for the preview](#).
- [SQL Server Management Studio \(SSMS\)](#)

## NOTE

During the public preview, Microsoft will onboard you and enable machine learning for your existing or new database.

This example uses the stored procedure [sp\\_execute\\_external\\_script](#) to wrap a well-formed R script.

## Run a simple script

To run an R script, you'll pass it as an argument to the system stored procedure, [sp\\_execute\\_external\\_script](#).

In the following steps, you'll run this example R script in your SQL database:

```
a <- 1
b <- 2
c <- a/b
d <- a*b
print(c(c, d))
```

1. Open **SQL Server Management Studio** and connect to your SQL database.

If you need help connecting, see [Quickstart: Use SQL Server Management Studio to connect and query an Azure SQL database](#).

2. Pass the complete R script to the [sp\\_execute\\_external\\_script](#) stored procedure.

The script is passed through the `@script` argument. Everything inside the `@script` argument must be valid R code.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
a <- 1
b <- 2
c <- a/b
d <- a*b
print(c(c, d))
'
```

If you get any errors, it might be because the public preview of Machine Learning Services (with R) is not enabled for your SQL database. See [Prerequisites](#) above.

#### NOTE

If you're an administrator, you can run external code automatically. You can grant permission to other users using the command:

**GRANT EXECUTE ANY EXTERNAL SCRIPT TO <username>.**

3. The correct result is calculated and the R `print` function returns the result to the **Messages** window.

It should look something like this.

## Results

```
STDOUT message(s) from external script:
0.5 2
```

## Run a Hello World script

A typical example script is one that just outputs the string "Hello World". Run the following command.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'OutputDataSet<-InputDataSet'
 , @input_data_1 = N'SELECT 1 AS hello'
WITH RESULT SETS(([Hello World] INT));
GO
```

Inputs to this stored procedure include:

@language	defines the language extension to call, in this case, R
@script	defines the commands passed to the R runtime. Your entire R script must be enclosed in this argument, as Unicode text. You could also add the text to a variable of type <b>nvarchar</b> and then call the variable
@input_data_1	data returned by the query, passed to the R runtime, which returns the data to SQL Server as a data frame
WITH RESULT SETS	clause defines the schema of the returned data table for SQL Server, adding "Hello World" as the column name, <b>int</b> for the data type

The command outputs the following text:

## Use inputs and outputs

By default, `sp_execute_external_script` accepts a single dataset as input, which typically you supply in the form of a valid SQL query. It then returns a single R data frame as output.

For now, let's use the default input and output variables of `sp_execute_external_script`: **InputDataSet** and **OutputDataSet**.

1. Create a small table of test data.

```
CREATE TABLE RTestData (col1 INT NOT NULL)

INSERT INTO RTestData
VALUES (1);

INSERT INTO RTestData
VALUES (10);

INSERT INTO RTestData
VALUES (100);
GO
```

2. Use the `SELECT` statement to query the table.

```
SELECT *
FROM RTestData
```

### Results

	col1
1	1
2	10
3	100

3. Run the following R script. It retrieves the data from the table using the `SELECT` statement, passes it through the R runtime, and returns the data as a data frame. The `WITH RESULT SETS` clause defines the schema of the returned data table for SQL Database, adding the column name `NewColumnName`.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'OutputDataSet <- InputDataSet;'
 , @input_data_1 = N'SELECT * FROM RTestData;'
WITH RESULT SETS(([NewColumnName] INT NOT NULL));
```

### Results

	NewColumnName
1	1
2	10
3	100

4. Now let's change the names of the input and output variables. The default input and output variable names are **InputDataSet** and **OutputDataSet**, this script changes the names to **SQL\_in** and **SQL\_out**:

```

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N' SQL_out <- SQL_in;'
 , @input_data_1 = N' SELECT 12 as Col;'
 , @input_data_1_name = N'SQL_in'
 , @output_data_1_name = N'SQL_out'
WITH RESULT SETS(([NewColName] INT NOT NULL));

```

Note that R is case-sensitive. The input and output variables used in the R script (**SQL\_out**, **SQL\_in**) need to match the values defined with `@input_data_1_name` and `@output_data_1_name`, including case.

#### TIP

Only one input dataset can be passed as a parameter, and you can return only one dataset. However, you can call other datasets from inside your R code and you can return outputs of other types in addition to the dataset. You can also add the OUTPUT keyword to any parameter to have it returned with the results.

5. You also can generate values just using the R script with no input data (`@input_data_1` is set to blank).

The following script outputs the text "hello" and "world".

```

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
mytextvariable <- c("hello", " ", "world");
OutputDataSet <- as.data.frame(mytextvariable);
'

 , @input_data_1 = N''
WITH RESULT SETS(([Col1] CHAR(20) NOT NULL));

```

#### Results

	Col1
1	hello
2	
3	world

## Check R version

If you would like to see which version of R is installed in your SQL database, run the following script.

```

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'print(version)';
GO

```

The R `print` function returns the version to the **Messages** window. In the example output below, you can see that SQL Database in this case has R version 3.4.4 installed.

#### Results

```
STDOUT message(s) from external script:
```

```
platform x86_64-w64-mingw32
arch x86_64
os mingw32
system x86_64, mingw32
status
major 3
minor 4.4
year 2018
month 03
day 15
svn rev 74408
language R
version.string R version 3.4.4 (2018-03-15)
nickname Someone to Lean On
```

## List R packages

Microsoft provides a number of R packages pre-installed with Machine Learning Services in your SQL database.

To see a list of which R packages are installed, including version, dependencies, license, and library path information, run the following script.

```
EXEC sp_execute_external_script @language = N'R'
 , @script = N'
OutputDataSet <- data.frame(installed.packages()[,c("Package", "Version", "Depends", "License", "LibPath")]);
WITH result sets(
 Package NVARCHAR(255)
 , Version NVARCHAR(100)
 , Depends NVARCHAR(4000)
 , License NVARCHAR(1000)
 , LibPath NVARCHAR(2000)
);'
```

The output is from `installed.packages()` in R and is returned as a result set.

### Results

	Package	Version	Depends	License	LibPath
1	abind	1.4-5	R (>= 1.5.0)	LGPL (>= 2)	C:/ExternalLibrariesSandboxPublicPath
2	assertthat	0.2.0	NULL	GPL-3	C:/ExternalLibrariesSandboxPublicPath
3	backports	1.1.0	R (>= 3.0.0)	GPL-2	C:/ExternalLibrariesSandboxPublicPath
4	base64enc	0.1-3	R (>= 2.9.0)	GPL-2   GPL-3	C:/ExternalLibrariesSandboxPublicPath
5	bdsmatrix	1.3-3	methods, R (>= 2.0.0)	LGPL-2	C:/ExternalLibrariesSandboxPublicPath
6	BH	1.66.0-1	NULL	BSL-1.0	C:/ExternalLibrariesSandboxPublicPath
7	bindr	0.1.1	NULL	MIT +file LICENSE	C:/ExternalLibrariesSandboxPublicPath
8	bindrcpp	0.2.2	NULL	MIT +file LICENSE	C:/ExternalLibrariesSandboxPublicPath
9	bit	1.1-14	R (>= 2.9.2)	GPL-2	C:/ExternalLibrariesSandboxPublicPath
10	bit64	0.9-7	R (>= 3.0.1), bit (>= 1.1-12), utils, methods, s...	GPL-2	C:/ExternalLibrariesSandboxPublicPath
11	bitops	1.0-6	NULL	GPL (>= 2)	C:/ExternalLibrariesSandboxPublicPath
12	blob	1.1.1	NULL	GPL-3	C:/ExternalLibrariesSandboxPublicPath
13	car	3.0-2	R (>= 3.2.0), carData (>= 3.0-0)	GPL (>= 2)	C:/ExternalLibrariesSandboxPublicPath
14	carData	3.0-1	R (>= 3.0)	GPL (>= 2)	C:/ExternalLibrariesSandboxPublicPath
15	caTools	1.17.1	R (>= 2.2.0)	GPL-3	C:/ExternalLibrariesSandboxPublicPath
16	cellranger	1.1.0	R (>= 3.0.0)	MIT +file LICENSE	C:/ExternalLibrariesSandboxPublicPath

## Next steps

To create a machine learning model using R in SQL Database, follow this quickstart:

[Create and train a predictive model in R with Azure SQL Database Machine Learning Services \(preview\)](#)

For more information on Azure SQL Database Machine Learning Services with R (preview), see the following articles.

- [Azure SQL Database Machine Learning Services with R \(preview\)](#)
- [Write advanced R functions in Azure SQL Database using Machine Learning Services \(preview\)](#)
- [Work with R and SQL data in Azure SQL Database Machine Learning Services \(preview\)](#)

# Quickstart: Create and train a predictive model in R with Azure SQL Database Machine Learning Services (preview)

1/26/2020 • 6 minutes to read • [Edit Online](#)

In this quickstart, you create and train a predictive model using R, save the model to a table in your database, then use the model to predict values from new data using Machine Learning Services (with R) in Azure SQL Database.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads.

Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An [Azure SQL database](#) with a [server-level firewall rule](#)
- [Machine Learning Services](#) with R enabled. [Sign up for the preview](#).
- [SQL Server Management Studio \(SSMS\)](#)

## NOTE

During the public preview, Microsoft will onboard you and enable machine learning for your existing or new database.

This example uses a simple regression model to predict the stopping distance of a car based on speed using the **cars** dataset included with R.

## TIP

Many datasets are included with the R runtime, to get a list of installed datasets, type `library(help="datasets")` from the R command prompt.

## Create and train a predictive model

The car speed data in the **cars** dataset contains two columns, both numeric: **dist** and **speed**. The data includes multiple stopping observations at different speeds. From this data, you'll create a linear regression model that describes the relationship between car speed and the distance required to stop a car.

The requirements of a linear model are simple:

- Define a formula that describes the relationship between the dependent variable *speed* and the independent variable *distance*.
- Provide input data to use in training the model.

**TIP**

If you need a refresher on linear models, try this tutorial which describes the process of fitting a model using rxLinMod:  
[Fitting Linear Models](#)

In the following steps you'll set up the training data, create a regression model, train it using the training data, then save the model to a SQL table.

1. Open **SQL Server Management Studio** and connect to your SQL database.

If you need help connecting, see [Quickstart: Use SQL Server Management Studio to connect and query an Azure SQL database](#).

2. Create the **CarSpeed** table to save the training data.

```
CREATE TABLE dbo.CarSpeed (
 speed INT NOT NULL
 , distance INT NOT NULL
)
GO

INSERT INTO dbo.CarSpeed (
 speed
 , distance
)
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'car_speed <- cars;'
 , @input_data_1 = N''
 , @output_data_1_name = N'car_speed'
GO
```

3. Create a regression model using `rxLinMod`.

To build the model you define the formula inside the R code and then pass the training data **CarSpeed** as an input parameter.

```
DROP PROCEDURE IF EXISTS generate_linear_model;
GO
CREATE PROCEDURE generate_linear_model
AS
BEGIN
 EXECUTE sp_execute_external_script
 @language = N'R'
 , @script = N'
lrmodel <- rxLinMod(formula = distance ~ speed, data = CarsData);
trained_model <- data.frame(payload = as.raw(serializers.serialize(lrmodel, connection=NULL)));
'

 , @input_data_1 = N'SELECT [speed], [distance] FROM CarSpeed'
 , @input_data_1_name = N'CarsData'
 , @output_data_1_name = N'trained_model'
 WITH RESULT SETS ((model VARBINARY(max)));
END;
GO
```

The first argument to rxLinMod is the *formula* parameter, which defines distance as dependent on speed. The input data is stored in the variable `CarsData`, which is populated by the SQL query.

4. Create a table where you store the model so you can use it later for prediction.

The output of an R package that creates a model is usually a **binary object**, so the table must have a

column of **VARBINARY(max)** type.

```
CREATE TABLE dbo.stopping_distance_models (
 model_name VARCHAR(30) NOT NULL DEFAULT('default model') PRIMARY KEY
 , model VARBINARY(max) NOT NULL
);
```

- Now call the stored procedure, generate the model, and save it to a table.

```
INSERT INTO dbo.stopping_distance_models (model)
EXECUTE generate_linear_model;
```

Note that if you run this code a second time, you get this error:

```
Violation of PRIMARY KEY constraint...Cannot insert duplicate key in object bo.stopping_distance_models
```

One option to avoid this error is to update the name for each new model. For example, you could change the name to something more descriptive, and include the model type, the day you created it, and so forth.

```
UPDATE dbo.stopping_distance_models
SET model_name = 'rxLinMod ' + FORMAT(GETDATE(), 'yyyy.MM.HH.mm', 'en-gb')
WHERE model_name = 'default model'
```

## View the table of coefficients

Generally, the output of R from the stored procedure [sp\\_execute\\_external\\_script](#) is limited to a single data frame. However, you can return outputs of other types, such as scalars, in addition to the data frame.

For example, suppose you want to train a model but immediately view the table of coefficients from the model. To do so, you create the table of coefficients as the main result set, and output the trained model in a SQL variable. You can immediately re-use the model by calling the variable, or you can save the model to a table as shown here.

```
DECLARE @model VARBINARY(max)
 , @modelname VARCHAR(30)

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
speedmodel <- rxLinMod(distance ~ speed, CarsData)
modelbin <- serialize(speedmodel, NULL)
OutputDataSet <- data.frame(coefficients(speedmodel));
'

 , @input_data_1 = N'SELECT [speed], [distance] FROM CarSpeed'
 , @input_data_1_name = N'CarsData'
 , @params = N'@modelbin varbinary(max) OUTPUT'
 , @modelbin = @model OUTPUT
WITH RESULT SETS(([Coefficient] FLOAT NOT NULL));

-- Save the generated model
INSERT INTO dbo.stopping_distance_models (
 model_name
 , model
)
VALUES (
 'latest model'
 , @model
)
```

## Results

	Coefficient
1	-17.579094890511
2	3.93240875912409

## Score new data using the trained model

*Scoring* is a term used in data science to mean generating predictions, probabilities, or other values based on new data fed into a trained model. You'll use the model you created in the previous section to score predictions against new data.

Did you notice that the original training data stops at a speed of 25 miles per hour? That's because the original data was based on an experiment from 1920! You might wonder, how long would it take an automobile from the 1920s to stop if it could get going as fast as 60 mph or even 100 mph? To answer this question, you can provide some new speed values to your model.

1. Create a table with new speed data.

```
CREATE TABLE dbo.NewCarSpeed (
 speed INT NOT NULL
 , distance INT NULL
)
GO

INSERT dbo.NewCarSpeed (speed)
VALUES (40)
 , (50)
 , (60)
 , (70)
 , (80)
 , (90)
 , (100)
```

2. Predict stopping distance from these new speed values.

Because your model is based on the **rxLinMod** algorithm provided as part of the **RevoScaleR** package, you call the **rxPredict** function, rather than the generic R `predict` function.

This example script:

- Uses a SELECT statement to get a single model from the table
- Passes it as an input parameter
- Calls the  `unserialize` function on the model
- Applies the  `rxPredict` function with appropriate arguments to the model
- Provides the new input data

### TIP

For real-time scoring, see [Serialization functions](#) provided by RevoScaleR.

```

DECLARE @speedmodel VARBINARY(max) =
 SELECT model
 FROM dbo.stopping_distance_models
 WHERE model_name = 'latest model'
);

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
current_model <- unserialize(as.raw(speedmodel));
new <- data.frame(NewCarData);
predicted.distance <- rxPredict(current_model, new);
str(predicted.distance);
OutputDataSet <- cbind(new, ceiling(predicted.distance));
'

 , @input_data_1 = N'SELECT speed FROM [dbo].[NewCarSpeed]'
 , @input_data_1_name = N'NewCarData'
 , @params = N'@speedmodel varbinary(max)'
 , @speedmodel = @speedmodel
WITH RESULT SETS(
 new_speed INT
 , predicted_distance INT
));

```

## Results

	new_speed	predicted_distance
1	40	140
2	50	180
3	60	219
4	70	258
5	80	298
6	90	337
7	100	376

### NOTE

In this example script, the `str` function is added during the testing phase to check the schema of data being returned from R. You can remove the statement later.

The column names used in the R script are not necessarily passed to the stored procedure output. Here the WITH RESULTS clause defines some new column names.

## Next steps

For more information on Azure SQL Database Machine Learning Services with R (preview), see the following articles.

- [Azure SQL Database Machine Learning Services with R \(preview\)](#)
- [Create and run simple R scripts in Azure SQL Database Machine Learning Services \(preview\)](#)
- [Write advanced R functions in Azure SQL Database using Machine Learning Services \(preview\)](#)
- [Work with R and SQL data in Azure SQL Database Machine Learning Services \(preview\)](#)

# Tutorial: Prepare data to train a predictive model in R with Azure SQL Database Machine Learning Services (preview)

7/28/2019 • 5 minutes to read • [Edit Online](#)

In part one of this three-part tutorial series, you'll import and prepare data from an Azure SQL database using R. Later in this series, you'll use this data to train and deploy a predictive machine learning model in R with Azure SQL Database Machine Learning Services (preview).

For this tutorial series, imagine you own a ski rental business and you want to predict the number of rentals that you'll have on a future date. This information will help you get your stock, staff, and facilities ready.

In parts one and two of this series, you'll develop some R scripts in RStudio to prepare your data and train a machine learning model. Then, in part three, you'll run those R scripts inside a SQL database using stored procedures.

In this article, you'll learn how to:

- Import a sample database into an Azure SQL database using R
- Load the data from the Azure SQL database into an R data frame
- Prepare the data in R by identifying some columns as categorical

In [part two](#), you'll learn how to create and train multiple machine learning models in R, and then choose the most accurate one.

In [part three](#), you'll learn how to store the model in a database, and then create stored procedures from the R scripts you developed in parts one and two. The stored procedures will run in a SQL database to make predictions based on new data.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads.

Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Azure subscription - If you don't have an Azure subscription, [create an account](#) before you begin.
- Azure SQL Database Server with Machine Learning Services enabled - During the public preview, Microsoft will onboard you and enable machine learning for your existing or new databases. Follow the steps in [Sign up for the preview](#).
- RevoScaleR package - See [RevoScaleR](#) for options to install this package locally.
- R IDE - This tutorial uses [RStudio Desktop](#).
- SQL query tool - This tutorial assumes you're using [Azure Data Studio](#) or [SQL Server Management Studio \(SSMS\)](#).

# Sign in to the Azure portal

Sign in to the [Azure portal](#).

## Import the sample database

The sample dataset used in this tutorial has been saved to a **.bacpac** database backup file for you to download and use.

1. Download the file [TutorialDB.bacpac](#).
2. Follow the directions in [Import a BACPAC file to create an Azure SQL database](#), using these details:
  - Import from the **TutorialDB.bacpac** file you downloaded
  - During the public preview, choose the **Gen5/vCore** configuration for the new database
  - Name the new database "TutorialDB"

## Load the data into a data frame

To use the data in R, you'll load the data from the Azure SQL database into a data frame (`rentaldata`).

Create a new RScript file in RStudio and run the following script. Replace **Server**, **UID**, and **PWD** with your own connection information.

```
#Define the connection string to connect to the TutorialDB database
connStr <- paste("Driver=SQL Server",
 "; Server=", "<Azure SQL Database Server>",
 "; Database=TutorialDB",
 "; UID=", "<user>",
 "; PWD=", "<password>",
 sep = "");

#Get the data from the table
SQL_rentaldata <- RxSqlServerData(table = "dbo.rental_data", connectionString = connStr, returnDataFrame =
TRUE);

#Import the data into a data frame
rentaldata <- rxImport(SQL_rentaldata);

#Take a look at the structure of the data and the top rows
head(rentaldata);
str(rentaldata);
```

You should see results similar to the following.

```
 Year Month Day RentalCount WeekDay Holiday Snow
1 2014 1 20 445 2 1 0
2 2014 2 13 40 5 0 0
3 2013 3 10 456 1 0 0
4 2014 3 31 38 2 0 0
5 2014 4 24 23 5 0 0
6 2015 2 11 42 4 0 0
'data.frame': 453 obs. of 7 variables:
 $ Year : int 2014 2014 2013 2014 2014 2015 2013 2014 2013 2015 ...
 $ Month : num 1 2 3 3 4 2 4 3 4 3 ...
 $ Day : num 20 13 10 31 24 11 28 8 5 29 ...
 $ RentalCount: num 445 40 456 38 23 42 310 240 22 360 ...
 $ WeekDay : num 2 5 1 2 5 4 1 7 6 1 ...
 $ Holiday : int 1 0 0 0 0 0 0 0 0 0 ...
 $ Snow : num 0 0 0 0 0 0 0 0 0 0 ...
```

## Prepare the data

In this sample database, most of the preparation has already been done, but you'll do one more preparation here. Use the following R script to identify three columns as *categories* by changing the data types to *factor*.

```
#Changing the three factor columns to factor types
rentaldata$Holiday <- factor(rentaldata$Holiday);
rentaldata$Snow <- factor(rentaldata$Snow);
rentaldata$WeekDay <- factor(rentaldata$WeekDay);

#Visualize the dataset after the change
str(rentaldata);
```

You should see results similar to the following.

```
data.frame': 453 obs. of 7 variables:
 $ Year : int 2014 2014 2013 2014 2014 2015 2013 2014 2013 2015 ...
 $ Month : num 1 2 3 3 4 2 4 3 4 3 ...
 $ Day : num 20 13 10 31 24 11 28 8 5 29 ...
 $ RentalCount: num 445 40 456 38 23 42 310 240 22 360 ...
 $ WeekDay : Factor w/ 7 levels "1","2","3","4",..: 2 5 1 2 5 4 1 7 6 1 ...
 $ Holiday : Factor w/ 2 levels "0","1": 2 1 1 1 1 1 1 1 1 1 ...
 $ Snow : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

The data is now prepared for training.

## Clean up resources

If you're not going to continue with this tutorial, delete the TutorialDB database from your Azure SQL Database server.

From the Azure portal, follow these steps:

1. From the left-hand menu in the Azure portal, select **All resources** or **SQL databases**.
2. In the **Filter by name...** field, enter **TutorialDB**, and select your subscription.
3. Select your TutorialDB database.
4. On the **Overview** page, select **Delete**.

## Next steps

In part one of this tutorial series, you completed these steps:

- Import a sample database into an Azure SQL database using R
- Load the data from the Azure SQL database into an R data frame
- Prepare the data in R by identifying some columns as categorical

To create a machine learning model that uses data from the TutorialDB database, follow part two of this tutorial series:

[Tutorial: Create a predictive model in R with Azure SQL Database Machine Learning Services \(preview\)](#)

# Tutorial: Create a predictive model in R with Azure SQL Database Machine Learning Services (preview)

7/28/2019 • 3 minutes to read • [Edit Online](#)

In part two of this three-part tutorial series, you'll create two predictive models in R and select the most accurate model. In the next part of this series, you'll deploy this model in a SQL database with Azure SQL Database Machine Learning Services (preview).

In this article, you'll learn how to:

- Train two machine learning models
- Make predictions from both models
- Compare the results to choose the most accurate model

In [part one](#), you learned how to import a sample database and then prepare the data to be used for training a predictive model in R.

In [part three](#), you'll learn how to store the model in a database, and then create stored procedures from the R scripts you developed in parts one and two. The stored procedures will run in a SQL database to make predictions based on new data.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Part two of this tutorial assumes you have completed [part one](#) and its prerequisites.

## Train two models

To find the best model for the ski rental data, create two different models (linear regression and decision tree) and see which one is predicting more accurately. You'll use the data frame `rentaldata` that you created in part one of this series.

```
#First, split the dataset into two different sets:
one for training the model and the other for validating it
train_data = rentaldata[rentaldata$Year < 2015,];
test_data = rentaldata[rentaldata$Year == 2015,];

#Use the RentalCount column to check the quality of the prediction against actual values
actual_counts <- test_data$RentalCount;

#Model 1: Use rxLinMod to create a linear regression model, trained with the training data set
model_linmod <- rxLinMod(RentalCount ~ Month + Day + WeekDay + Snow + Holiday, data = train_data);

#Model 2: Use rxDTree to create a decision tree model, trained with the training data set
model_dtree <- rxDTree(RentalCount ~ Month + Day + WeekDay + Snow + Holiday, data = train_data);
```

## Make predictions from both models

Use a predict function to predict the rental counts using each trained model.

```
#Use both models to make predictions using the test data set.
predict_linmod <- rxPredict(model_linmod, test_data, writeModelVars = TRUE, extraVarsToWrite = c("Year"));

predict_dtree <- rxPredict(model_dtree, test_data, writeModelVars = TRUE, extraVarsToWrite = c("Year"));

#To verify it worked, look at the top rows of the two prediction data sets.
head(predict_linmod);
head(predict_dtree);
```

	RentalCount_Pred	RentalCount	Month	Day	WeekDay	Snow	Holiday
1	27.45858	42	2	11	4	0	0
2	387.29344	360	3	29	1	0	0
3	16.37349	20	4	22	4	0	0
4	31.07058	42	3	6	6	0	0
5	463.97263	405	2	28	7	1	0
6	102.21695	38	1	12	2	1	0

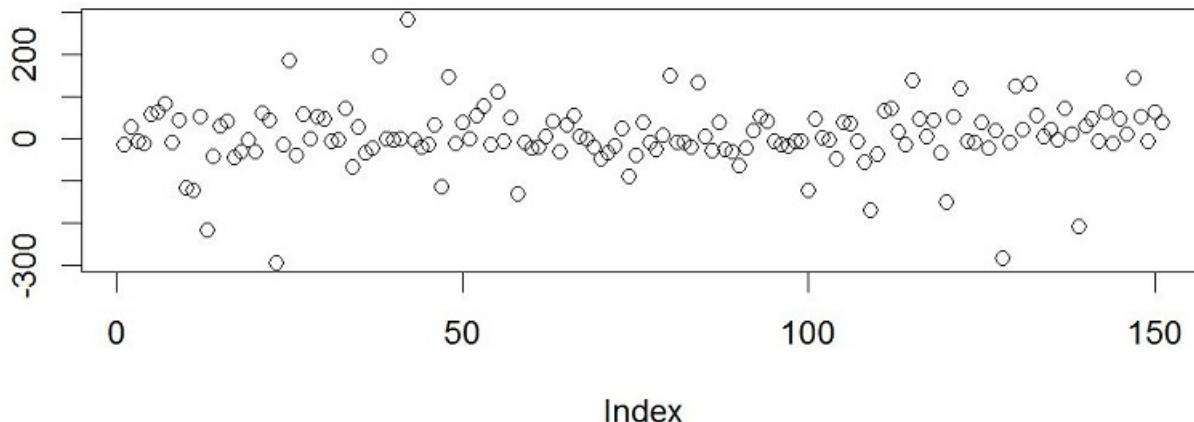
	RentalCount_Pred	RentalCount	Month	Day	WeekDay	Snow	Holiday
1	40.0000	42	2	11	4	0	0
2	332.5714	360	3	29	1	0	0
3	27.7500	20	4	22	4	0	0
4	34.2500	42	3	6	6	0	0
5	645.7059	405	2	28	7	1	0
6	40.0000	38	1	12	2	1	0

## Compare the results

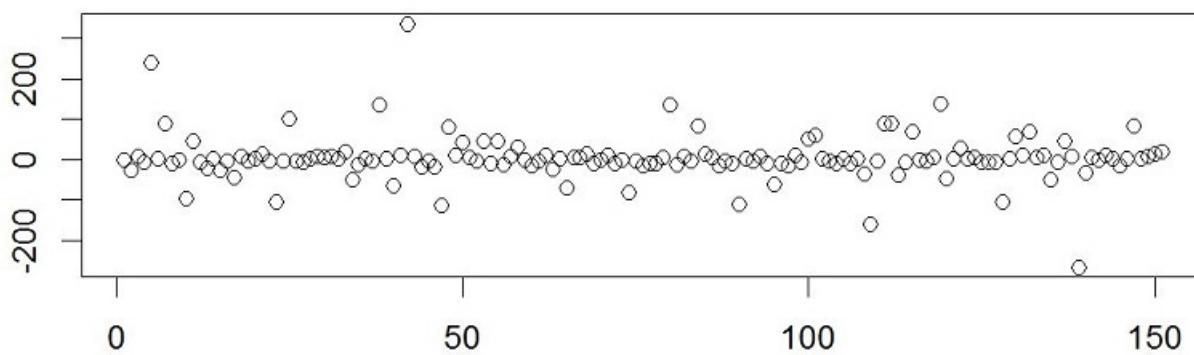
Now you want to see which of the models gives the best predictions. A quick and easy way to do this is to use a basic plotting function to view the difference between the actual values in your training data and the predicted values.

```
#Use the plotting functionality in R to visualize the results from the predictions
par(mfrow = c(2, 1));
plot(predict_linmod$RentalCount_Pred - predict_linmod$RentalCount, main = "Difference between actual and
predicted. rxLinmod");
plot(predict_dtree$RentalCount_Pred - predict_dtree$RentalCount, main = "Difference between actual and
predicted. rxDTree");
```

## Difference between actual and predicted. rxLinmod



## Difference between actual and predicted. rxDTree



It looks like the decision tree model is the more accurate of the two models.

## Clean up resources

If you're not going to continue with this tutorial, delete the TutorialDB database from your Azure SQL Database server.

From the Azure portal, follow these steps:

1. From the left-hand menu in the Azure portal, select **All resources** or **SQL databases**.
2. In the **Filter by name...** field, enter **TutorialDB**, and select your subscription.
3. Select your TutorialDB database.
4. On the **Overview** page, select **Delete**.

## Next steps

In part two of this tutorial series, you completed these steps:

- Train two machine learning models
- Make predictions from both models
- Compare the results to choose the most accurate model

To deploy the machine learning model you've created, follow part three of this tutorial series:



# Tutorial: Deploy a predictive model in R with Azure SQL Database Machine Learning Services (preview)

7/28/2019 • 4 minutes to read • [Edit Online](#)

In part three of this three-part tutorial, you'll deploy a predictive model, developed in R, into a SQL database using Azure SQL Database Machine Learning Services (preview).

You'll create a stored procedure with an embedded R script that makes predictions using the model. Because your model executes in the Azure SQL database, it can easily be trained against data stored in the database.

In this article, using the R scripts you developed in parts one and two, you'll learn how to:

- Create a stored procedure that generates the machine learning model
- Store the model in a database table
- Create a stored procedure that makes predictions using the model
- Execute the model with new data

In [part one](#), you learned how to import a sample database and then prepare the data to be used for training a predictive model in R.

In [part two](#), you learned how to create and train multiple machine learning models in R, and then choose the most accurate one.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads.

Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Part three of this tutorial series assumes you have completed [part one](#) and [part two](#).

## Create a stored procedure that generates the model

In part two of this tutorial series, you decided that a decision tree (dtree) model was the most accurate. Now, using the R scripts you developed, create a stored procedure (`generate_rental_rx_model`) that trains and generates the dtree model using rxDTree from the RevoScaleR package.

Run the following commands in Azure Data Studio or SSMS.

```

-- Stored procedure that trains and generates an R model using the rental_data and a decision tree algorithm
DROP PROCEDURE IF EXISTS generate_rental_rx_model;
GO
CREATE PROCEDURE generate_rental_rx_model (@trained_model VARBINARY(max) OUTPUT)
AS
BEGIN
 EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
require("RevoScaleR");

rental_train_data$Holiday <- factor(rental_train_data$Holiday);
rental_train_data$Snow <- factor(rental_train_data$Snow);
rental_train_data$WeekDay <- factor(rental_train_data$WeekDay);

#Create a dtree model and train it using the training data set
model_dtree <- rxDTTree(RentalCount ~ Month + Day + WeekDay + Snow + Holiday, data = rental_train_data);
#Serialize the model before saving it to the database table
trained_model <- as.raw(serialize(model_dtree, connection=NULL));
'

 , @input_data_1 = N'
 SELECT RentalCount
 , Year
 , Month
 , Day
 , WeekDay
 , Snow
 , Holiday
 FROM dbo.rental_data
 WHERE Year < 2015
 '

 , @input_data_1_name = N'rental_train_data'
 , @params = N'@trained_model varbinary(max) OUTPUT'
 , @trained_model = @trained_model OUTPUT;
END;
GO

```

## Store the model in a database table

Create a table in the TutorialDB database and then save the model to the table.

1. Create a table (`rental_rx_models`) for storing the model.

```

USE TutorialDB;
DROP TABLE IF EXISTS rental_rx_models;
GO
CREATE TABLE rental_rx_models (
 model_name VARCHAR(30) NOT NULL DEFAULT('default model') PRIMARY KEY
 , model VARBINARY(MAX) NOT NULL
);
GO

```

2. Save the model to the table as a binary object, with the model name "rxDTree".

```
-- Save model to table
TRUNCATE TABLE rental_rx_models;

DECLARE @model VARBINARY(MAX);

EXECUTE generate_rental_rx_model @model OUTPUT;

INSERT INTO rental_rx_models (
 model_name
 , model
)
VALUES (
 'rxDTree'
 , @model
);

SELECT *
FROM rental_rx_models;
```

## Create a stored procedure that makes predictions

Create a stored procedure (`predict_rentalcount_new`) that makes predictions using the trained model and a set of new data.

```
-- Stored procedure that takes model name and new data as input parameters and predicts the rental count for
the new data
DROP PROCEDURE IF EXISTS predict_rentalcount_new;
GO
CREATE PROCEDURE predict_rentalcount_new (
 @model_name VARCHAR(100)
 , @input_query NVARCHAR(MAX)
)
AS
BEGIN
 DECLARE @rx_model VARBINARY(MAX) = (
 SELECT model
 FROM rental_rx_models
 WHERE model_name = @model_name
);

 EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
require("RevoScaleR");

#Convert types to factors
rentals$Holiday <- factor(rentals$Holiday);
rentals$Snow <- factor(rentals$Snow);
rentals$WeekDay <- factor(rentals$WeekDay);

#Before using the model to predict, we need to unserialize it
rental_model <- unserialize(rx_model);

#Call prediction function
rental_predictions <- rxPredict(rental_model, rentals);
'

 , @input_data_1 = @input_query
 , @input_data_1_name = N'rentals'
 , @output_data_1_name = N'rental_predictions'
 , @params = N'@rx_model varbinary(max)'
 , @rx_model = @rx_model
 WITH RESULT SETS(("RentalCount_Predicted" FLOAT));
END;
GO
```

## Execute the model with new data

Now you can use the stored procedure `predict_rentalcount_new` to predict the rental count from new data.

```
-- Use the predict_rentalcount_new stored procedure with the model name and a set of features to predict the
rental count
EXECUTE dbo.predict_rentalcount_new @model_name = 'rxDTree'
 , @input_query =
 SELECT CONVERT(INT, 3) AS Month
 , CONVERT(INT, 24) AS Day
 , CONVERT(INT, 4) AS WeekDay
 , CONVERT(INT, 1) AS Snow
 , CONVERT(INT, 1) AS Holiday
 ;
GO
```

You should see a result similar to the following.

```
RentalCount_Predicted
332.571428571429
```

You have successfully created, trained, and deployed a model in an Azure SQL database. You then used that model in a stored procedure to predict values based on new data.

## Clean up resources

When you've finished using the TutorialDB database, delete it from your Azure SQL Database server.

From the Azure portal, follow these steps:

1. From the left-hand menu in the Azure portal, select **All resources** or **SQL databases**.
2. In the **Filter by name...** field, enter **TutorialDB**, and select your subscription.
3. Select your TutorialDB database.
4. On the **Overview** page, select **Delete**.

## Next steps

In part three of this tutorial series, you completed these steps:

- Create a stored procedure that generates the machine learning model
- Store the model in a database table
- Create a stored procedure that makes predictions using the model
- Execute the model with new data

To learn more about using R in Azure SQL Database Machine Learning Services (preview), see:

- [Write advanced R functions in Azure SQL Database using Machine Learning Services \(preview\)](#)
- [Work with R and SQL data in Azure SQL Database Machine Learning Services \(preview\)](#)
- [Add an R package to Azure SQL Database Machine Learning Services \(preview\)](#)

# Tutorial: Prepare data to perform clustering in R with Azure SQL Database Machine Learning Services (preview)

7/30/2019 • 5 minutes to read • [Edit Online](#)

In part one of this three-part tutorial series, you'll import and prepare the data from an Azure SQL database using R. Later in this series, you'll use this data to train and deploy a clustering model in R with Azure SQL Database Machine Learning Services (preview).

*Clustering* can be explained as organizing data into groups where members of a group are similar in some way. You'll use the **K-Means** algorithm to perform the clustering of customers in a dataset of product purchases and returns. By clustering customers, you can focus your marketing efforts more effectively by targeting specific groups. K-Means clustering is an *unsupervised learning* algorithm that looks for patterns in data based on similarities.

In parts one and two of this series, you'll develop some R scripts in RStudio to prepare your data and train a machine learning model. Then, in part three, you'll run those R scripts inside a SQL database using stored procedures.

In this article, you'll learn how to:

- Import a sample database into an Azure SQL database
- Separate customers along different dimensions using R
- Load the data from the Azure SQL database into an R data frame

In [part two](#), you'll learn how to create and train a K-Means clustering model in R.

In [part three](#), you'll learn how to create a stored procedure in an Azure SQL database that can perform clustering in R based on new data.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads.

Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Azure subscription - If you don't have an Azure subscription, [create an account](#) before you begin.
- Azure SQL Database Server with Machine Learning Services enabled - During the public preview, Microsoft will onboard you and enable machine learning for your existing or new databases. Follow the steps in [Sign up for the preview](#).
- RevoScaleR package - See [RevoScaleR](#) for options to install this package locally.
- R IDE - This tutorial uses [RStudio Desktop](#).
- SQL query tool - This tutorial assumes you're using [Azure Data Studio](#) or [SQL Server Management Studio](#)

(SSMS).

# Sign in to the Azure portal

Sign in to the [Azure portal](#).

# Import the sample database

The sample dataset used in this tutorial has been saved to a **.bacpac** database backup file for you to download and use. This dataset is derived from the [tpcx-bb](#) dataset provided by the [Transaction Processing Performance Council \(TPC\)](#).

1. Download the file [tpcxbb\\_1gb.bacpac](#).
  2. Follow the directions in [Import a BACPAC file to create an Azure SQL database](#), using these details:
    - Import from the **tpcxbb\_1gb.bacpac** file you downloaded
    - During the public preview, choose the **Gen5/vCore** configuration for the new database
    - Name the new database "tpcxbb\_1gb"

## Separate customers

Create a new RScript file in RStudio and run the following script. In the SQL query, you're separating customers along the following dimensions:

- **orderRatio** = return order ratio (total number of orders partially or fully returned versus the total number of orders)
  - **itemsRatio** = return item ratio (total number of items returned versus the number of items purchased)
  - **monetaryRatio** = return amount ratio (total monetary amount of items returned versus the amount purchased)
  - **frequency** = return frequency

In the **Paste** function, replace **Server**, **UID**, and **PWD** with your own connection information.

```

Define the connection string to connect to the tpcxbb_1gb database
connStr <- paste("Driver=SQL Server",
 "; Server=", "<Azure SQL Database Server>",
 "; Database=tpcxbb_1gb",
 "; UID=", "<user>",
 "; PWD=", "<password>",
 sep = "");

```

---

```

#Define the query to select data from SQL Server
input_query <- "
SELECT ss_customer_sk AS customer
,round(CASE
WHEN (
 (orders_count = 0)
 OR (returns_count IS NULL)
 OR (orders_count IS NULL)
 OR ((returns_count / orders_count) IS NULL)
)
THEN 0.0
ELSE (cast(returns_count AS NCHAR(10)) / orders_count)
END, 7) AS orderRatio
,round(CASE
WHEN (
 (orders_items = 0)
 OR (returns_items IS NULL)
 OR (orders_items IS NULL)
)

```

```

 OR (orders_items IS NULL)
 OR ((returns_items / orders_items) IS NULL)
)
THEN 0.0
ELSE (cast(returns_items AS NCHAR(10)) / orders_items)
END, 7) AS itemsRatio
,round(CASE
WHEN (
 (orders_money = 0)
 OR (returns_money IS NULL)
 OR (orders_money IS NULL)
 OR ((returns_money / orders_money) IS NULL)
)
THEN 0.0
ELSE (cast(returns_money AS NCHAR(10)) / orders_money)
END, 7) AS monetaryRatio
,round(CASE
WHEN (returns_count IS NULL)
THEN 0.0
ELSE returns_count
END, 0) AS frequency
FROM (
SELECT ss_customer_sk,
-- return order ratio
COUNT(DISTINCT (ss_ticket_number)) AS orders_count,
-- return ss_item_sk ratio
COUNT(ss_item_sk) AS orders_items,
-- return monetary amount ratio
SUM(ss_net_paid) AS orders_money
FROM store_sales s
GROUP BY ss_customer_sk
) orders
LEFT OUTER JOIN (
SELECT sr_customer_sk,
-- return order ratio
count(DISTINCT (sr_ticket_number)) AS returns_count,
-- return ss_item_sk ratio
COUNT(sr_item_sk) AS returns_items,
-- return monetary amount ratio
SUM(sr_return_amt) AS returns_money
FROM store_returns
GROUP BY sr_customer_sk
) returned ON ss_customer_sk = sr_customer_sk
"

```

## Load the data into a data frame

Now use the following script to return the results from the query to an R data frame using the **rxSqlServerData** function. As part of the process, you'll define the type for the selected columns (using colClasses) to make sure that the types are correctly transferred to R.

```

Query SQL Server using input_query and get the results back
to data frame customer_returns
Define the types for selected columns (using colClasses),
to make sure that the types are correctly transferred to R
customer_returns <- rxSqlServerData(
 sqlQuery=input_query,
 colClasses=c(customer ="numeric",
 orderRatio="numeric",
 itemsRatio="numeric",
 monetaryRatio="numeric",
 frequency="numeric"),
 connectionString=connStr);

Transform the data from an input dataset to an output dataset
customer_data <- rxDataStep(customer_returns);

Take a look at the data just loaded from SQL Server
head(customer_data, n = 5);

```

You should see results similar to the following.

	customer	orderRatio	itemsRatio	monetaryRatio	frequency
1	29727	0	0	0.000000	0
2	26429	0	0	0.041979	1
3	60053	0	0	0.065762	3
4	97643	0	0	0.037034	3
5	32549	0	0	0.031281	4

## Clean up resources

If you're not going to continue with this tutorial, delete the tpcxbb\_1gb database from your Azure SQL Database server.

From the Azure portal, follow these steps:

1. From the left-hand menu in the Azure portal, select **All resources** or **SQL databases**.
2. In the **Filter by name...** field, enter **tpcxxb\_1gb**, and select your subscription.
3. Select your **tpcxxb\_1gb** database.
4. On the **Overview** page, select **Delete**.

## Next steps

In part one of this tutorial series, you completed these steps:

- Import a sample database into an Azure SQL database
- Separate customers along different dimensions using R
- Load the data from the Azure SQL database into an R data frame

To create a machine learning model that uses this customer data, follow part two of this tutorial series:

[Tutorial: Create a predictive model in R with Azure SQL Database Machine Learning Services \(preview\)](#)

# Tutorial: Build a clustering model in R with Azure SQL Database Machine Learning Services (preview)

7/30/2019 • 4 minutes to read • [Edit Online](#)

In part two of this three-part tutorial series, you'll build a K-Means model in R to perform clustering. In the next part of this series, you'll deploy this model in a SQL database with Azure SQL Database Machine Learning Services (preview).

In this article, you'll learn how to:

- Define the number of clusters for a K-Means algorithm
- Perform clustering
- Analyze the results

In [part one](#), you learned how to prepare the data from an Azure SQL database to perform clustering.

In [part three](#), you'll learn how to create a stored procedure in an Azure SQL database that can perform clustering in R based on new data.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Part two of this tutorial assumes you have completed [part one](#) and its prerequisites.

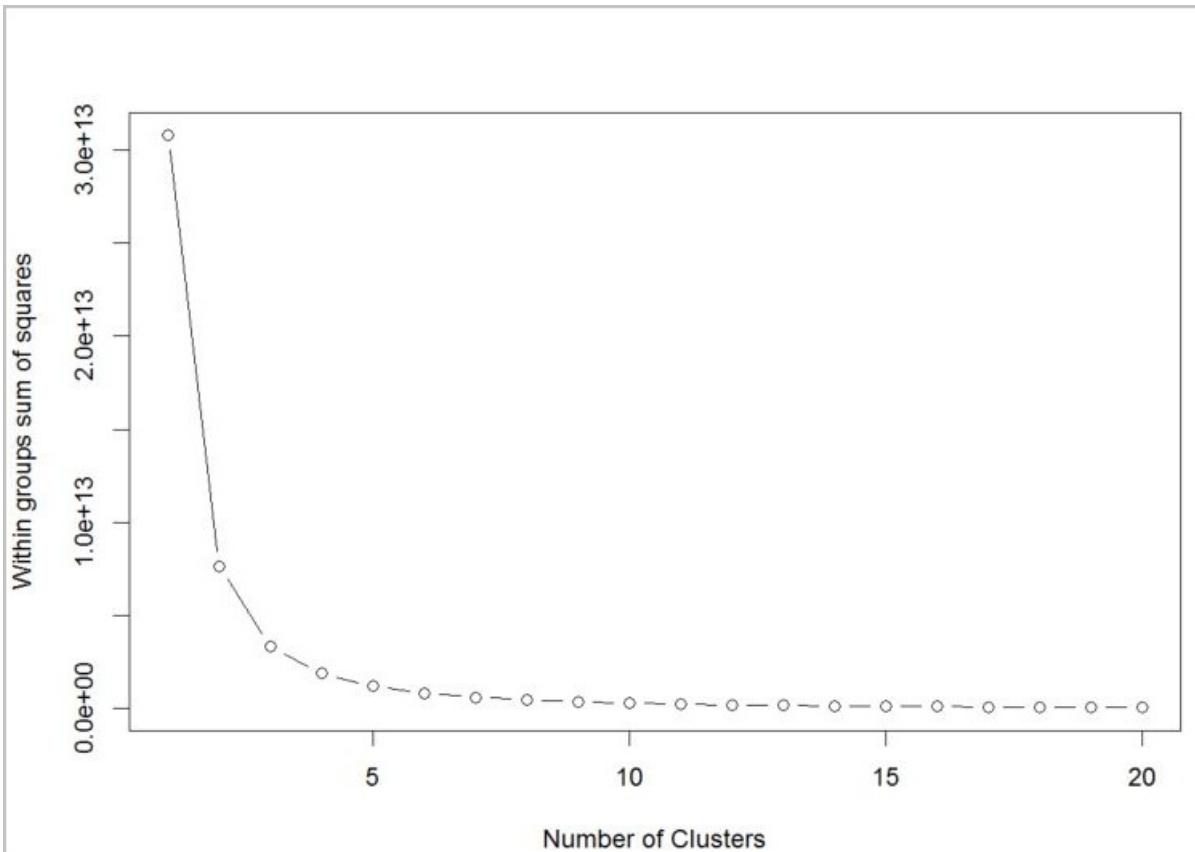
## Define the number of clusters

To cluster your customer data, you'll use the **K-Means** clustering algorithm, one of the simplest and most well-known ways of grouping data. You can read more about K-Means in [A complete guide to K-means clustering algorithm](#).

The algorithm accepts two inputs: The data itself, and a predefined number " $k$ " representing the number of clusters to generate. The output is  $k$  clusters with the input data partitioned among the clusters.

To determine the number of clusters for the algorithm to use, use a plot of the within groups sum of squares, by number of clusters extracted. The appropriate number of clusters to use is at the bend or "elbow" of the plot.

```
Determine number of clusters by using a plot of the within groups sum of squares,
by number of clusters extracted.
wss <- (nrow(customer_data) - 1) * sum(apply(customer_data, 2, var))
for (i in 2:20)
 wss[i] <- sum(kmeans(customer_data, centers = i)$withinss)
plot(1:20, wss, type = "b", xlab = "Number of Clusters", ylab = "Within groups sum of squares")
```



Based on the graph, it looks like  $k = 4$  would be a good value to try. That  $k$  value will group the customers into four clusters.

## Perform clustering

In the following R script, you'll use the function **rxKmeans**, which is the K-Means function in the RevoScaleR package.

```
Output table to hold the customer group mappings.
This is a table where the cluster mappings will be saved in the database.
return_cluster = RxSqlServerData(table = "return_cluster", connectionString = connStr);
Set the seed for the random number generator for predictability
set.seed(10);
Generate clusters using rxKmeans and output key / cluster to a table in SQL database
called return_cluster
clust <- rxKmeans(~ orderRatio + itemsRatio + monetaryRatio + frequency,
 customer_returns,
 numClusters=4,
 outFile=return_cluster,
 outColName="cluster",
 extraVarsToWrite=c("customer"),
 overwrite=TRUE);

Read the customer returns cluster table from the database
customer_cluster <- rxDataStep(return_cluster);
```

## Analyze the results

Now that you've done the clustering using K-Means, the next step is to analyze the result and see if you can find any actionable information.

The **clust** object contains the results from the K-Means clustering.

```
#Look at the clustering details to analyze results
clust
```

```
Call:
rxKmeans(formula = ~orderRatio + itemsRatio + monetaryRatio +
 frequency, data = customer_returns, outFile = return_cluster,
 outColName = "cluster", extraVarsToWrite = c("customer"),
 overwrite = TRUE, numClusters = 4)
Data: customer_returns
Number of valid observations: 37336
Number of missing observations: 0
Clustering algorithm:

K-means clustering with 4 clusters of sizes 31675, 671, 2851, 2139
Cluster means:
 orderRatio itemsRatio monetaryRatio frequency
1 0.000000000 0.000000000 0.0000000 0.000000
2 0.007451565 0.000000000 0.04449653 4.271237
3 1.008067345 0.2707821817 0.49515232 1.031568
4 0.000000000 0.0004675082 0.10858272 1.186068
Within cluster sum of squares by cluster:
 1 2 3 4
0.0000 1329.0160 18561.3157 363.2188
```

The four cluster means are given using the variables defined in [part one](#):

- *orderRatio* = return order ratio (total number of orders partially or fully returned versus the total number of orders)
- *itemsRatio* = return item ratio (total number of items returned versus the number of items purchased)
- *monetaryRatio* = return amount ratio (total monetary amount of items returned versus the amount purchased)
- *frequency* = return frequency

Data mining using K-Means often requires further analysis of the results, and further steps to better understand each cluster, but it can provide some good leads. Here are a couple ways you could interpret these results:

- Cluster 1 (the largest cluster) seems to be a group of customers that are not active (all values are zero).
- Cluster 3 seems to be a group that stands out in terms of return behavior.

## Clean up resources

If you're not going to continue with this tutorial, delete the tpcxbb\_1gb database from your Azure SQL Database server.

From the Azure portal, follow these steps:

1. From the left-hand menu in the Azure portal, select **All resources** or **SQL databases**.
2. In the **Filter by name...** field, enter **tpcxbb\_1gb**, and select your subscription.
3. Select your **tpcxbb\_1gb** database.
4. On the **Overview** page, select **Delete**.

## Next steps

In part two of this tutorial series, you completed these steps:

- Define the number of clusters for a K-Means algorithm
- Perform clustering
- Analyze the results

To deploy the machine learning model you've created, follow part three of this tutorial series:

[Tutorial: Deploy a clustering model in R with Azure SQL Database Machine Learning Services \(preview\)](#)

# Tutorial: Deploy a clustering model in R with Azure SQL Database Machine Learning Services (preview)

7/30/2019 • 5 minutes to read • [Edit Online](#)

In part three of this three-part tutorial series, you'll deploy a clustering model, developed in R, into a SQL database using Azure SQL Database Machine Learning Services (preview).

You'll create a stored procedure with an embedded R script that performs clustering. Because your model executes in the Azure SQL database, it can easily be trained against data stored in the database.

In this article, you'll learn how to:

- Create a stored procedure that generates the model
- Perform clustering in SQL Database
- Use the clustering information

In [part one](#), you learned how to prepare the data from an Azure SQL database to perform clustering.

In [part two](#), you learned how to create and train a K-Means clustering model in R.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Part three of this tutorial series assumes you have completed [part one](#) and [part two](#).

## Create a stored procedure that generates the model

Run the following T-SQL script to create the stored procedure. The procedure recreates the steps you developed in parts one and two of this tutorial series:

- classify customers based on their purchase and return history
- generate four clusters of customers using a K-Means algorithm

The procedure stores the resulting customer cluster mappings in the database table **customer\_return\_clusters**.

```
USE [tpcxbb_1gb]
DROP PROC IF EXISTS generate_customer_return_clusters;
GO
CREATE procedure [dbo].[generate_customer_return_clusters]
AS
/*
 This procedure uses R to classify customers into different groups
 based on their purchase & return history.
*/
BEGIN
 DECLARE @duration FLOAT
 . @instance name NVARCHAR(100) = @@SERVERNAME
```

```

 , @instance_name NVARCHAR(128) = db_name()
 , @database_name NVARCHAR(128) = db_name()
-- Input query to generate the purchase history & return metrics
 , @input_query NVARCHAR(MAX) = N'
SELECT ss_customer_sk AS customer,
round(CASE
 WHEN (
 (orders_count = 0)
 OR (returns_count IS NULL)
 OR (orders_count IS NULL)
 OR ((returns_count / orders_count) IS NULL)
)
 THEN 0.0
 ELSE (cast(returns_count AS NCHAR(10)) / orders_count)
 END, 7) AS orderRatio,
round(CASE
 WHEN (
 (orders_items = 0)
 OR (returns_items IS NULL)
 OR (orders_items IS NULL)
 OR ((returns_items / orders_items) IS NULL)
)
 THEN 0.0
 ELSE (cast(returns_items AS NCHAR(10)) / orders_items)
 END, 7) AS itemsRatio,
round(CASE
 WHEN (
 (orders_money = 0)
 OR (returns_money IS NULL)
 OR (orders_money IS NULL)
 OR ((returns_money / orders_money) IS NULL)
)
 THEN 0.0
 ELSE (cast(returns_money AS NCHAR(10)) / orders_money)
 END, 7) AS monetaryRatio,
round(CASE
 WHEN (returns_count IS NULL)
 THEN 0.0
 ELSE returns_count
 END, 0) AS frequency
FROM (
 SELECT ss_customer_sk,
 -- return order ratio
 COUNT(DISTINCT (ss_ticket_number)) AS orders_count,
 -- return ss_item_sk ratio
 COUNT(ss_item_sk) AS orders_items,
 -- return monetary amount ratio
 SUM(ss_net_paid) AS orders_money
 FROM store_sales s
 GROUP BY ss_customer_sk
) orders
LEFT OUTER JOIN (
 SELECT sr_customer_sk,
 -- return order ratio
 COUNT(DISTINCT (sr_ticket_number)) AS returns_count,
 -- return ss_item_sk ratio
 COUNT(sr_item_sk) AS returns_items,
 -- return monetary amount ratio
 SUM(sr_return_amt) AS returns_money
 FROM store_returns
 GROUP BY sr_customer_sk
) returned ON ss_customer_sk = sr_customer_sk
'

EXECUTE sp_execute_external_script
 @language = N'R'
 , @script = N'
Define the connection string
connStr <- paste("Driver=SQL Server; Server=", instance_name,
 "; Database=", database_name,
 "; Trusted_Connection=true; "

```

```

 , trusted_Connection=True,
 sep="");

Input customer data that needs to be classified.
This is the result we get from the query.
customer_returns <- RxSqlServerData(
 sqlQuery=input_query,
 colClasses=c(customer ="numeric",
 orderRatio="numeric",
 itemsRatio="numeric",
 monetaryRatio="numeric",
 frequency="numeric"),
 connectionString=connStr);
Output table to hold the customer cluster mappings
return_cluster = RxSqlServerData(table = "customer_return_clusters",
 connectionString = connStr);

Set seed for random number generator for predictability
set.seed(10);

Generate clusters using rxKmeans and output clusters to a table
called "customer_return_clusters"
clust <- rxKmeans(~ orderRatio + itemsRatio + monetaryRatio + frequency,
 customer_returns,
 numClusters = 4,
 outFile = return_cluster,
 outColName = "cluster",
 writeModelVars = TRUE ,
 extraVarsToWrite = c("customer"),
 overwrite = TRUE);
'

, @input_data_1 = N'
, @params = N'@instance_name nvarchar(100), @database_name nvarchar(128), @input_query nvarchar(max),
@duration float OUTPUT'
, @instance_name = @instance_name
, @database_name = @database_name
, @input_query = @input_query
, @duration = @duration OUTPUT;
END;

GO

```

## Perform clustering in SQL Database

Now that you've created the stored procedure, execute the following script to perform clustering.

```

--Empty table of the results before running the stored procedure
TRUNCATE TABLE customer_return_clusters;

--Execute the clustering
--This will load the table customer_return_clusters with cluster mappings
EXECUTE [dbo].[generate_customer_return_clusters];

```

Verify that it works and that we actually have the list of customers and their cluster mappings.

```

--Select data from table customer_return_clusters
--to verify that the clustering data was loaded
SELECT TOP (5) *
FROM customer_return_clusters;

```

cluster	customer	orderRatio	itemsRatio	monetaryRatio	frequency
1	29727	0	0	0	0
4	26429	0	0	0.041979	1
2	60053	0	0	0.065762	3
2	97643	0	0	0.037034	3
2	32549	0	0	0.031281	4

## Use the clustering information

Because you stored the clustering procedure in the database, it can perform clustering efficiently against customer data stored in the same database. You can execute the procedure whenever your customer data is updated and use the updated clustering information.

Suppose you want to send a promotional email to customers in cluster 3, the group that has more active return behavior (you can see how the four clusters were described in [part two](#)). The following code selects the email addresses of customers in cluster 3.

```
USE [tpcxxbb_1gb]

SELECT customer.[c_email_address],
 customer.c_customer_sk
 FROM dbo.customer
 JOIN [dbo].[customer_return_clusters] AS r ON r.customer = customer.c_customer_sk
 WHERE r.cluster = 3
```

You can change the **r.cluster** value to return email addresses for customers in other clusters.

## Clean up resources

When you're finished with this tutorial, you can delete the tpcxbb\_1gb database from your Azure SQL Database server.

From the Azure portal, follow these steps:

1. From the left-hand menu in the Azure portal, select **All resources** or **SQL databases**.
2. In the **Filter by name...** field, enter **tpcxxbb\_1gb**, and select your subscription.
3. Select your **tpcxxbb\_1gb** database.
4. On the **Overview** page, select **Delete**.

## Next steps

In part three of this tutorial series, you completed these steps:

- Create a stored procedure that generates the model
- Perform clustering in SQL Database
- Use the clustering information

To learn more about using R in Azure SQL Database Machine Learning Services (preview), see:

- [Tutorial: Prepare data to train a predictive model in R with Azure SQL Database Machine Learning Services \(preview\)](#)
- [Write advanced R functions in Azure SQL Database using Machine Learning Services \(preview\)](#)
- [Work with R and SQL data in Azure SQL Database Machine Learning Services \(preview\)](#)
- [Add an R package to Azure SQL Database Machine Learning Services \(preview\)](#)

# Write advanced R functions in Azure SQL Database using Machine Learning Services (preview)

4/21/2019 • 3 minutes to read • [Edit Online](#)

This article describes how to embed R mathematical and utility functions in a SQL stored procedure. Advanced statistical functions that are complicated to implement in T-SQL can be done in R with only a single line of code.

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- If you don't have an Azure subscription, [create an account](#) before you begin.
- To run the example code in these exercises, you must first have an Azure SQL database with Machine Learning Services (with R) enabled. During the public preview, Microsoft will onboard you and enable machine learning for your existing or new database. Follow the steps in [Sign up for the preview](#).
- Make sure you've installed the latest [SQL Server Management Studio](#) (SSMS). You can run R scripts using other database management or query tools, but in this quickstart you'll use SSMS.

## Create a stored procedure to generate random numbers

For simplicity, let's use the R `stats` package that's installed and loaded by default with Azure SQL Database using Machine Learning Services (preview). The package contains hundreds of functions for common statistical tasks, among them the `rnorm` function. This function generates a specified number of random numbers using the normal distribution, given a standard deviation and means.

For example, the following R code returns 100 numbers on a mean of 50, given a standard deviation of 3.

```
as.data.frame(rnorm(100, mean = 50, sd = 3));
```

To call this line of R from T-SQL, run `sp_execute_external_script` and add the R function in the R script parameter, like this:

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
OutputDataSet <- as.data.frame(rnorm(100, mean = 50, sd =3));
'
 , @input_data_1 = N' ;'
WITH RESULT SETS(([Density] FLOAT NOT NULL));
```

What if you'd like to make it easier to generate a different set of random numbers?

That's easy when combined with SQL. You define a stored procedure that gets the arguments from the user, then

pass those arguments into the R script as variables.

```
CREATE PROCEDURE MyRNorm (
 @param1 INT
 , @param2 INT
 , @param3 INT
)
AS
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
OutputDataSet <- as.data.frame(rnorm(mynumbers, mymean, mysd));
'

 , @input_data_1 = N' ;'
 , @params = N' @mynumbers int, @mymean int, @mysd int'
 , @mynumbers = @param1
 , @mymean = @param2
 , @mysd = @param3
WITH RESULT SETS(([Density] FLOAT NOT NULL));
```

- The first line defines each of the SQL input parameters that are required when the stored procedure is executed.
- The line beginning with `@params` defines all variables used by the R code, and the corresponding SQL data types.
- The lines that immediately follow map the SQL parameter names to the corresponding R variable names.

Now that you've wrapped the R function in a stored procedure, you can easily call the function and pass in different values, like this:

```
EXECUTE MyRNorm @param1 = 100
 , @param2 = 50
 , @param3 = 3
```

## Use R utility functions for troubleshooting

The `utils` package, installed by default, provides a variety of utility functions for investigating the current R environment. These functions can be useful if you're finding discrepancies in the way your R code performs in SQL and in outside environments. For example, you might use the R `memory.limit()` function to get memory for the current R environment.

Because the `utils` package is installed but not loaded by default, you must use the `library()` function to load it first.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
library(utils);
mymemory <- memory.limit();
OutputDataSet <- as.data.frame(mymemory);

 , @input_data_1 = N' ;'
WITH RESULT SETS(([Col1] INT NOT NULL));
```

### TIP

Many users like to use the system timing functions in R, such as `system.time` and `proc.time`, to capture the time used by R processes and analyze performance issues.



# Work with R and SQL data in Azure SQL Database Machine Learning Services (preview)

1/23/2020 • 9 minutes to read • [Edit Online](#)

This article discusses some of the common issues you may encounter when moving data between R and SQL Database in [Machine Learning Services \(with R\) in Azure SQL Database](#). The experience you gain through this exercise provides essential background when working with data in your own script.

Common issues that you may encounter include:

- Data types sometimes don't match
- Implicit conversions might take place
- Cast and convert operations are sometimes required
- R and SQL use different data objects

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- If you don't have an Azure subscription, [create an account](#) before you begin.
- To run the example code in these exercises, you must first have an Azure SQL database with Machine Learning Services (with R) enabled. During the public preview, Microsoft will onboard you and enable machine learning for your existing or new database. Follow the steps in [Sign up for the preview](#).
- Make sure you've installed the latest [SQL Server Management Studio](#) (SSMS). You can run R scripts using other database management or query tools, but in this quickstart you'll use SSMS.

## Working with a data frame

When your script returns results from R to SQL, it must return the data as a **data.frame**. Any other type of object that you generate in your script - whether that be a list, factor, vector, or binary data - must be converted to a data frame if you want to output it as part of the stored procedure results. Fortunately, there are multiple R functions to support changing other objects to a data frame. You can even serialize a binary model and return it in a data frame, which you'll do later in this article.

First, let's experiment with some basic R objects - vectors, matrices, and lists - and see how conversion to a data frame changes the output passed to SQL.

Compare these two "Hello World" scripts in R. The scripts look almost identical, but the first returns a single column of three values, whereas the second returns three columns with a single value each.

### Example 1

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N' mytextvariable <- c("hello", " ", "world");
OutputDataSet <- as.data.frame(mytextvariable);
'
 , @input_data_1 = N'';
```

## Example 2

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N' OutputDataSet<- data.frame(c("hello"), " ", c("world"));
 , @input_data_1 = N'';
```

Why are the results so different?

The answer can usually be found by using the R `str()` command. Add the function `str(object_name)` anywhere in your R script to have the data schema of the specified R object returned as an informational message. You can view the messages in the **Messages** tab in SSMS.

To figure out why Example 1 and Example 2 have such different results, insert the line `str(OutputDataSet)` at the end of the `@script` variable definition in each statement, like this:

### Example 1 with str function added

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
mytextvariable <- c("hello", " ", "world");
OutputDataSet <- as.data.frame(mytextvariable);
str(OutputDataSet);
'
 , @input_data_1 = N' ';
```

### Example 2 with str function added

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
OutputDataSet <- data.frame(c("hello"), " ", c("world"));
str(OutputDataSet);
'
 , @input_data_1 = N' ';
```

Now, review the text in **Messages** to see why the output is different.

### Results - Example 1

```
STDOUT message(s) from external script:
'data.frame': 3 obs. of 1 variable:
$ mytextvariable: Factor w/ 3 levels " ","hello","world": 2 1 3
```

### Results - Example 2

```
STDOUT message(s) from external script:
'data.frame': 1 obs. of 3 variables:
$ c..hello..: Factor w/ 1 level "hello": 1
$ X... : Factor w/ 1 level " ": 1
$ c..world..: Factor w/ 1 level "world": 1
```

As you can see, a slight change in R syntax had a big effect on the schema of the results. For all the details, the differences in R data types are explained in details in the *Data Structures* section in "[Advanced R](#)" by Hadley Wickham.

For now, just be aware that you need to check the expected results when coercing R objects into data frames.

**TIP**

You also can use R identity functions, such as `is.matrix`, `is.vector`, to return information about the internal data structure.

## Implicit conversion of data objects

Each R data object has its own rules for how values are handled when combined with other data objects if the two data objects have the same number of dimensions, or if any data object contains heterogeneous data types.

For example, assume you want to perform matrix multiplication using R. You want to multiply a single-column matrix with the three values by an array with four values, and expect a 4x3 matrix as a result.

First, create a small table of test data.

```
CREATE TABLE RTestData (col1 INT NOT NULL)

INSERT INTO RTestData
VALUES (1);

INSERT INTO RTestData
VALUES (10);

INSERT INTO RTestData
VALUES (100);
GO
```

Now run the following script.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
x <- as.matrix(InputDataSet);
y <- array(12:15);
OutputDataSet <- as.data.frame(x %*% y);
'

, @input_data_1 = N' SELECT [Col1] from RTestData;'
WITH RESULT SETS((
 [Col1] INT
 , [Col2] INT
 , [Col3] INT
 , Col4 INT
));
```

Under the covers, the column of three values is converted to a single-column matrix. Because a matrix is just a special case of an array in R, the array `y` is implicitly coerced to a single-column matrix to make the two arguments conform.

### Results

COL1	COL2	COL3	COL4
12	13	14	15

COL1	COL2	COL3	COL4
------	------	------	------

120	130	140	150
1200	1300	1400	1500

However, note what happens when you change the size of the array `y`.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
x <- as.matrix(InputDataSet);
y <- array(12:14);
OutputDataSet <- as.data.frame(y %%*% x);
'

, @input_data_1 = N' SELECT [Col1] from RTestData;
WITH RESULT SETS(([Col1] INT));
```

Now R returns a single value as the result.

## Results

COL1
1542

Why? In this case, because the two arguments can be handled as vectors of the same length, R returns the inner product as a matrix. This is the expected behavior according to the rules of linear algebra. However, it could cause problems if your downstream application expects the output schema to never change!

## Merge or multiply columns of different length

R provides great flexibility for working with vectors of different sizes, and for combining these column-like structures into data frames. Lists of vectors can look like a table, but they don't follow all the rules that govern database tables.

For example, the following script defines a numeric array of length 6 and stores it in the R variable `df1`. The numeric array is then combined with the integers of the RTestData table (created above) which contains three (3) values, to make a new data frame, `df2`.

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
df1 <- as.data.frame(array(1:6));
df2 <- as.data.frame(c(InputDataSet , df1));
OutputDataSet <- df2;

'

, @input_data_1 = N' SELECT [Col1] from RTestData;
WITH RESULT SETS(
 [Col2] INT NOT NULL
 , [Col3] INT NOT NULL
);
```

To fill out the data frame, R repeats the elements retrieved from RTestData as many times as needed to match the number of elements in the array `df1`.

## Results

<b>COL2</b>	<b>COL3</b>
1	1
10	2
100	3
1	4
10	5
100	6

Remember that a data frame only looks like a table, but is actually a list of vectors.

## Cast or convert SQL data

R and SQL don't use the same data types, so when you run a query in SQL to get data and then pass that to the R runtime, some type of implicit conversion usually takes place. Another set of conversions takes place when you return data from R to SQL.

- SQL pushes the data from the query to the R process and converts it to an internal representation for greater efficiency.
- The R runtime loads the data into a data.frame variable and performs its own operations on the data.
- The database engine returns the data to SQL using a secured internal connection and presents the data in terms of SQL data types.
- You get the data by connecting to SQL using a client or network library that can issue SQL queries and handle tabular data sets. This client application can potentially affect the data in other ways.

To see how this works, run a query such as this one on the [AdventureWorksDW](#) data warehouse. This view returns sales data used in creating forecasts.

```
USE AdventureWorksDW
GO

SELECT ReportingDate
 , CAST(ModelRegion AS VARCHAR(50)) AS ProductSeries
 , Amount
 FROM [AdventureWorksDW].[dbo].[vTimeSeries]
 WHERE [ModelRegion] = 'M200 Europe'
 ORDER BY ReportingDate ASC
```

### NOTE

You can use any version of AdventureWorks, or create a different query using a database of your own. The point is to try to handle some data that contains text, datetime, and numeric values.

Now, try using this query as the input to the stored procedure.

```

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
str(InputDataSet);
OutputDataSet <- InputDataSet;

, @input_data_1 = N'
 SELECT ReportingDate
 , CAST(ModelRegion as varchar(50)) as ProductSeries
 , Amount
 FROM [AdventureWorksDW].[dbo].[vTimeSeries]
 WHERE [ModelRegion] = ''M200 Europe''
 ORDER BY ReportingDate ASC ;
'
WITH RESULT SETS undefined;

```

If you get an error, you'll probably need to make some edits to the query text. For example, the string predicate in the WHERE clause must be enclosed by two sets of single quotation marks.

After you get the query working, review the results of the `str` function to see how R treats the input data.

## Results

```

STDOUT message(s) from external script: 'data.frame': 37 obs. of 3 variables:
STDOUT message(s) from external script: $ ReportingDate: POSIXct, format: "2010-12-24 23:00:00" "2010-12-24
23:00:00"
STDOUT message(s) from external script: $ ProductSeries: Factor w/ 1 levels "M200 Europe",...: 1 1 1 1 1 1 1
1 1
STDOUT message(s) from external script: $ Amount : num 3400 16925 20350 16950 16950

```

- The datetime column has been processed using the R data type, **POSIXct**.
- The text column "ProductSeries" has been identified as a **factor**, meaning a categorical variable. String values are handled as factors by default. If you pass a string to R, it is converted to an integer for internal use, and then mapped back to the string on output.

## Summary

From even these short examples, you can see the need to check the effects of data conversion when passing SQL queries as input. Because some SQL data types are not supported by R, consider these ways to avoid errors:

- Test your data in advance and verify columns or values in your schema that could be a problem when passed to R code.
- Specify columns in your input data source individually, rather than using `SELECT *`, and know how each column will be handled.
- Perform explicit casts as necessary when preparing your input data, to avoid surprises.
- Avoid passing columns of data (such as GUIDS or rowguids) that cause errors and aren't useful for modeling.

For more information on supported and unsupported R data types, see [R libraries and data types](#).

# Add an R package to Azure SQL Database Machine Learning Services (preview)

11/7/2019 • 4 minutes to read • [Edit Online](#)

This article explains how to add an R package to Azure SQL Database Machine Learning Services (preview).

## IMPORTANT

Azure SQL Database Machine Learning Services is currently in preview. Sign up for the preview is currently closed.

This preview version is provided without a service level agreement, and it's not recommended for production workloads. Certain features might not be supported or might have constrained capabilities. For more information, see [Supplemental Terms of Use for Microsoft Azure Previews](#).

## Prerequisites

- Install [R](#) and [RStudio Desktop](#) on your local computer. R is available for Windows, MacOS, and Linux. This article assumes you're using Windows.
- This article includes an example of using [Azure Data Studio](#) or [SQL Server Management Studio \(SSMS\)](#) to run an R script in Azure SQL Database. You can run R scripts using other database management or query tools, but this example assumes Azure Data Studio or SSMS.

## NOTE

You can't install a package by running an R script using `sp_execute_external_script` in Azure Data Studio or SSMS. You can only install and remove packages using the R command line and RStudio as described in this article. Once the package is installed, you can access the package functions in an R script using `sp_execute_external_script`.

## List R packages

Microsoft provides a number of R packages pre-installed with Machine Learning Services in your Azure SQL database. You can see a list of installed R packages by running the following command in Azure Data Studio or SSMS.

1. Open Azure Data Studio or SSMS and connect to your Azure SQL Database.
2. Run the following command:

```
EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
OutputDataSet <- data.frame(installed.packages()[,c("Package", "Version", "Depends", "License")]);'
WITH RESULT SETS(
 Package NVARCHAR(255)
 , Version NVARCHAR(100)
 , Depends NVARCHAR(4000)
 , License NVARCHAR(1000)
)';
```

The output should look similar to the following.

## Results

Package	Version	Depends	License
base	3.4.4	NULL	Part of R 3.4.4
boot	1.3-20	R (>= 3.0.0), graphics, stats	Unlimited
checkpoint	0.4.3	R(>= 3.0.0)	GPL-2
class	7.3-14	R (>= 3.0.0), stats, utils	GPL-2   GPL-3
cluster	2.0.6	R (>= 3.0.1)	GPL (>= 2)
codetools	0.2-15	R (>= 2.11)	GPL

## Add a package with **sqlmlutils**

If you need to use a package that isn't already installed in your Azure SQL Database, you can install it using **sqlmlutils**. **sqlmlutils** is a package designed to help users interact with SQL databases (SQL Server and Azure SQL Database) and execute R or Python code in SQL from an R or Python client. Currently, only the R version of **sqlmlutils** is supported in Azure SQL Database.

In the following example, you'll install the **glue** package that can format and interpolate strings. These steps install **sqlmlutils** and **RODBCext** (a prerequisite for **sqlmlutils**), and add the **glue** package.

### Install **sqlmlutils**

1. Download the latest **sqlmlutils** zip file from <https://github.com/Microsoft/sqlmlutils/tree/master/R/dist> to your local computer. You don't need to unzip the file.
2. Open a **Command Prompt** and run the following commands to install **RODBCext** and **sqlmlutils** on your local computer. Substitute the full path to the **sqlmlutils** zip file you downloaded (the example assumes the file is in your Documents folder).

```
R -e "install.packages('RODBCext', repos='https://cran.microsoft.com')"
R CMD INSTALL %UserProfile%\Documents\sqlmlutils_0.5.1.zip
```

The output you see should be similar to the following.

```
In R CMD INSTALL
* installing to library 'C:/Users/<username>/Documents/R/win-library/3.5'
package sqlmlutils successfully unpacked and MD5 sums checked
```

### TIP

If you get the error, "'R' is not recognized as an internal or external command, operable program or batch file", it likely means that the path to R.exe is not included in your **PATH** environment variable on Windows. You can either add the path to the environment variable or navigate to the folder in the command prompt (for example `cd C:\Program Files\R\R-3.5.3\bin`) and then retry the command.

### Add the package

1. Open RStudio and create a new **R Script** file.
2. Use the following R code to install the **glue** package using **sqlmlutils**. Substitute your own Azure SQL Database connection information.

```

library(sqlmlutils)
connection <- connectionInfo(
 server= "yourserver.database.windows.net",
 database = "yourdatabase",
 uid = "yoursqluser",
 pwd = "yoursqlpassword")

sql_install.packages(connectionString = connection, pkgs = "glue", verbose = TRUE, scope = "PUBLIC")

```

### TIP

The **scope** can be either **PUBLIC** or **PRIVATE**. Public scope is useful for the database administrator to install packages that all users can use. Private scope makes the package available only to the user who installs it. If you don't specify the scope, the default scope is **PRIVATE**.

## Verify the package

Verify that the **glue** package has been installed by running the following R script in RStudio. Use the same **connection** you defined in the previous step.

```
r<-sql_installed.packages(connectionString = connection, fields=c("Package", "Version", "Depends", "License"))
View(r)
```

## Results

Package	Version	Depends	License
glue	glue	R (>= 3.1)	MIT + file LICENSE
base	base	NA	Part of R 3.4.4
boot	boot	R (>= 3.0.0), graphics, stats	Unlimited
checkpoint	checkpoint	R(>= 3.0.0)	CPL-2

## Use the package

Once the package is installed, you can use it in an R script through **sp\_execute\_external\_script**.

1. Open Azure Data Studio or SSMS and connect to your Azure SQL Database.
2. Run the following command:

```

EXECUTE sp_execute_external_script @language = N'R'
 , @script = N'
library(glue)

name <- "Fred"
age <- 50
anniversary <- as.Date("2020-06-14")
text <- glue(''My name is {name}, '',
'my age next year is {age + 1}, '',
'my anniversary is {format(anniversary, "%A, %B %d, %Y")}'.'')

print(text)
';

```

You'll see the following result in the **Messages** tab.

## Results

My name is Fred, my age next year is 51, my anniversary is Sunday, June 14, 2020.

## Remove the package

If you would like to remove the package, run the following R script in RStudio. Use the same **connection** you defined earlier.

```
sql_remove.packages(connectionString = connection, pkgs = "glue", scope = "PUBLIC")
```

### TIP

Another way to install an R package to your Azure SQL database is to upload the R package from a byte stream using the **CREATE EXTERNAL LIBRARY** T-SQL statement. See [Create a library from a byte stream](#) in the **CREATE EXTERNAL LIBRARY** reference documentation.

## Next steps

For more information about Azure SQL Database Machine Learning Services with R (preview), see the following articles.

- [Azure SQL Database Machine Learning Services with R \(preview\)](#)
- [Write advanced R functions in Azure SQL Database using Machine Learning Services \(preview\)](#)
- [Work with R and SQL data in Azure SQL Database Machine Learning Services \(preview\)](#)

# How to use a single database in Azure SQL Database

1/15/2020 • 2 minutes to read • [Edit Online](#)

In this section you can find various guides, scripts, and explanations that can help you to manage and configure your single database in Azure SQL Database

## Migrate

- [Migrate to SQL Database](#) – Learn about the recommended migration process and tools for migration to a managed instance.
- Learn how to [manage SQL database after migration](#).

## Configure features

- [Configure transactional replication](#) to replicate your date between databases.
- [Configure threat detection](#) to let Azure SQL Database identify suspicious activities such as SQL Injection or access from suspicious locations.
- [Configure dynamic data masking](#) to protect your sensitive data.
- [Configure backup retention](#) for a database to keep your backups on Azure Blob Storage. As an alternative there is [Configure backup retention using Azure vault \(deprecated\)](#) approach.
- [Configure geo-replication](#) to keep a replica of your database in another region.
- [Configure security for geo-replicas](#).

## Monitor and tune your database

- [Enable automatic tuning](#) to let Azure SQL Database optimize performance of your workload.
- [Enable e-mail notifications for automatic tuning](#) to get information about tuning recommendations.
- [Apply performance recommendations](#) and optimize your database.
- [Create alerts](#) to get notifications from Azure SQL Database.
- [Troubleshoot connectivity](#) if you notice some connectivity issues between the applications and the database.  
You can also use [Resource Health for connectivity issues](#).
- [Manage file space](#) to monitor storage usage in your database.

## Query distributed data

- [Query vertically partitioned data](#) across multiple databases.
- [Report across scaled-out data tier](#).
- [Query across tables with different schemas](#).

## Elastic Database Jobs

- [Create and manage](#) Elastic Database Jobs using PowerShell.
- [Create and manage](#) Elastic Database Jobs using Transact-SQL.
- [Migrate from old Elastic job](#).

## Database sharding

- [Upgrade elastic database client library](#).

- [Create sharded app.](#)
- [Query horizontally sharded data.](#)
- [Run Multi-shard queries.](#)
- [Move sharded data.](#)
- [Configure security](#) in database shards.
- [Add a shard](#) to the current set of database shards.
- [Fix shard map problems.](#)
- [Migrate sharded DB.](#)
- [Create counters.](#)
- [Use entity framework](#) to query sharded data.
- [Use Dapper framework](#) to query sharded data.

## Next steps

- Learn more about [How-to guides for managed instance](#)

# SQL Server database migration to Azure SQL Database

1/14/2020 • 6 minutes to read • [Edit Online](#)

In this article, you learn about the primary methods for migrating a SQL Server 2005 or later database to a single or pooled database in Azure SQL Database. For information on migrating to a Managed Instance, see [Migrate to SQL Server instance to Azure SQL Database Managed Instance](#). For migration information about migrating from other platforms, see [Azure Database Migration Guide](#).

## Migrate to a single database or a pooled database

There are two primary methods for migrating a SQL Server 2005 or later database to a single or pooled database in Azure SQL Database. The first method is simpler but requires some, possibly substantial, downtime during the migration. The second method is more complex, but substantially eliminates downtime during the migration.

In both cases, you need to ensure that the source database is compatible with Azure SQL Database using the [Data Migration Assistant \(DMA\)](#). SQL Database V12 is approaching [feature parity](#) with SQL Server, other than issues related to server-level and cross-database operations. Databases and applications that rely on [partially supported or unsupported functions](#) need some [re-engineering to fix these incompatibilities](#) before the SQL Server database can be migrated.

### NOTE

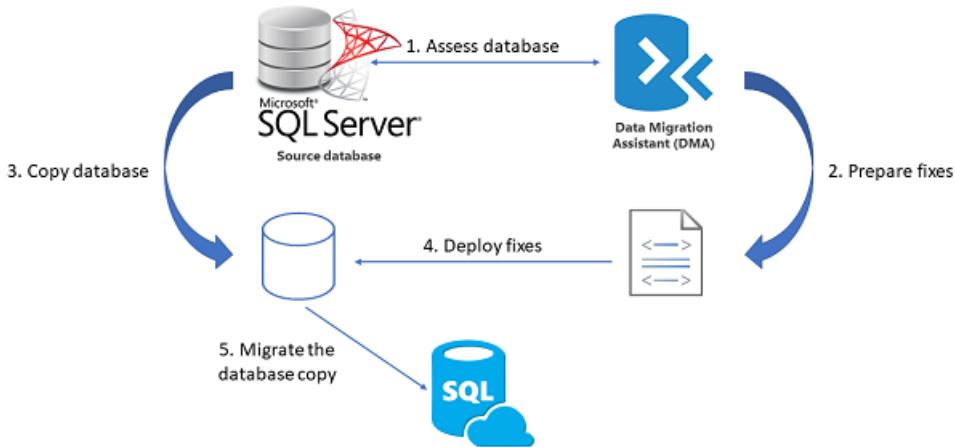
To migrate a non-SQL Server database, including Microsoft Access, Sybase, MySQL Oracle, and DB2 to Azure SQL Database, see [SQL Server Migration Assistant](#).

## Method 1: Migration with downtime during the migration

Use this method to migrate to a single or a pooled database if you can afford some downtime or you are performing a test migration of a production database for later migration. For a tutorial, see [Migrate a SQL Server database](#).

The following list contains the general workflow for a SQL Server database migration of a single or a pooled database using this method. For migration to Managed Instance, see [Migration to a Managed Instance](#).

# Azure SQL Database migration



1. [Assess](#) the database for compatibility by using the latest version of the [Data Migration Assistant \(DMA\)](#).
2. Prepare any necessary fixes as Transact-SQL scripts.
3. Make a transactionally consistent copy of the source database being migrated or halt new transactions from occurring in the source database while migration is occurring. Methods to accomplish this latter option include disabling client connectivity or creating a [database snapshot](#). After migration, you may be able to use transactional replication to update the migrated databases with changes that occur after the cutoff point for the migration. See [Migrate using Transactional Migration](#).
4. Deploy the Transact-SQL scripts to apply the fixes to the database copy.
5. [Migrate](#) the database copy to a new Azure SQL Database by using the Data Migration Assistant.

#### NOTE

Rather than using DMA, you can also use a BACPAC file. See [Import a BACPAC file to a new Azure SQL Database](#).

## Optimizing data transfer performance during migration

The following list contains recommendations for best performance during the import process.

- Choose the highest service tier and compute size that your budget allows to maximize the transfer performance. You can scale down after the migration completes to save money.
- Minimize the distance between your BACPAC file and the destination data center.
- Disable auto-statistics during migration
- Partition tables and indexes
- Drop indexed views, and recreate them once finished
- Remove rarely queried historical data to another database and migrate this historical data to a separate Azure SQL database. You can then query this historical data using [elastic queries](#).

## Optimize performance after the migration completes

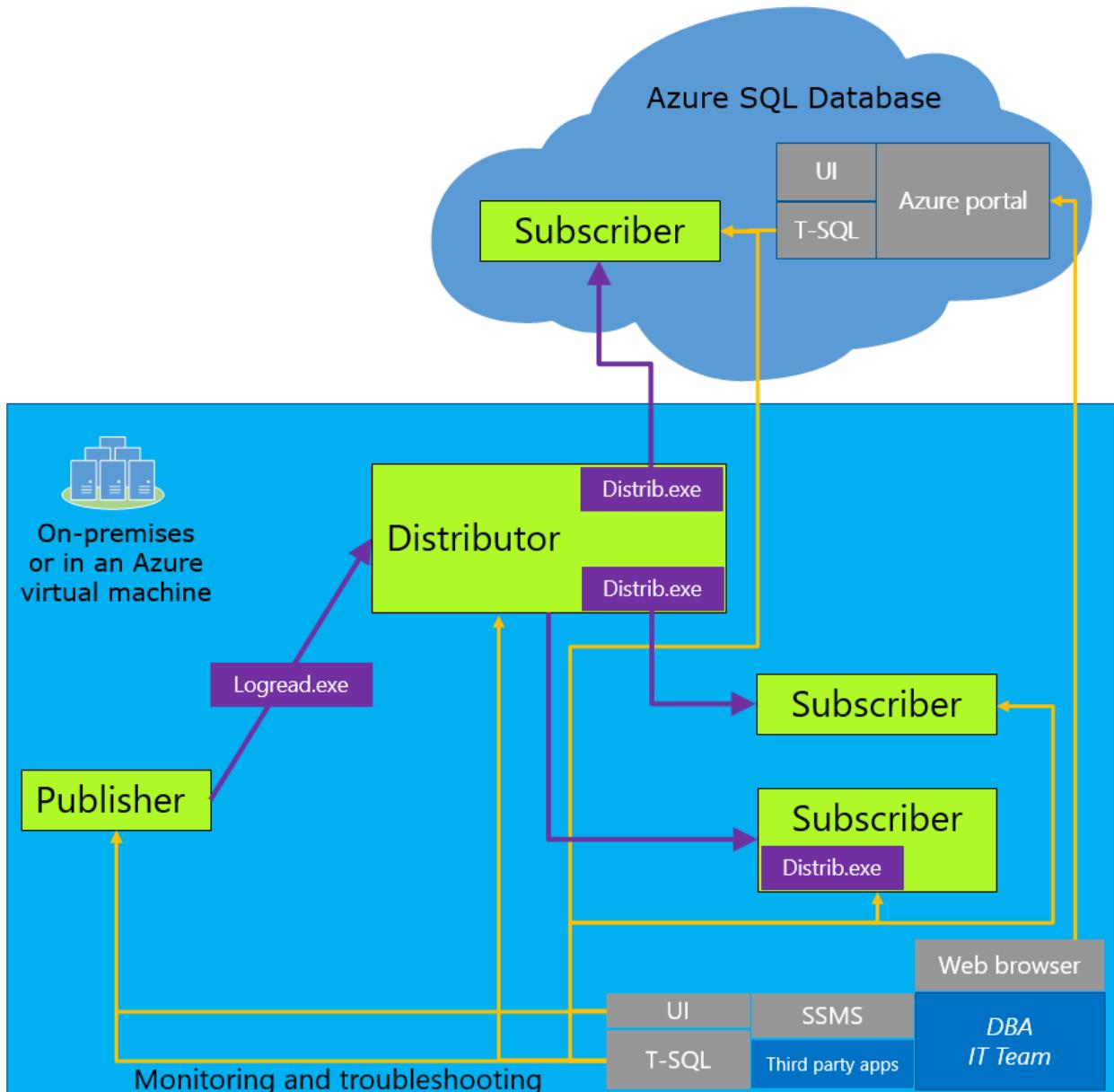
[Update statistics](#) with full scan after the migration is completed.

## Method 2: Use Transactional Replication

When you cannot afford to remove your SQL Server database from production while the migration is occurring, you can use SQL Server transactional replication as your migration solution. To use this method, the source database must meet the [requirements for transactional replication](#) and be compatible for Azure SQL Database. For information about SQL replication with Always On, see [Configure Replication for Always On Availability Groups \(SQL Server\)](#).

To use this solution, you configure your Azure SQL Database as a subscriber to the SQL Server instance that you wish to migrate. The transactional replication distributor synchronizes data from the database to be synchronized (the publisher) while new transactions continue occur.

With transactional replication, all changes to your data or schema show up in your Azure SQL Database. Once the synchronization is complete and you are ready to migrate, change the connection string of your applications to point them to your Azure SQL Database. Once transactional replication drains any changes left on your source database and all your applications point to Azure DB, you can uninstall transactional replication. Your Azure SQL Database is now your production system.



**TIP**

You can also use transactional replication to migrate a subset of your source database. The publication that you replicate to Azure SQL Database can be limited to a subset of the tables in the database being replicated. For each table being replicated, you can limit the data to a subset of the rows and/or a subset of the columns.

## Migration to SQL Database using Transaction Replication workflow

**IMPORTANT**

Use the latest version of SQL Server Management Studio to remain synchronized with updates to Microsoft Azure and SQL Database. Older versions of SQL Server Management Studio cannot set up SQL Database as a subscriber. [Update SQL Server Management Studio](#).

## 1. Set up Distribution

- [Using SQL Server Management Studio \(SSMS\)](#)
- [Using Transact-SQL](#)

## 2. Create Publication

- [Using SQL Server Management Studio \(SSMS\)](#)
- [Using Transact-SQL](#)

## 3. Create Subscription

- [Using SQL Server Management Studio \(SSMS\)](#)
- [Using Transact-SQL](#)

Some tips and differences for migrating to SQL Database

- Use a local distributor
  - Doing so causes a performance impact on the server.
  - If the performance impact is unacceptable, you can use another server but it adds complexity in management and administration.
- When selecting a snapshot folder, make sure the folder you select is large enough to hold a BCP of every table you want to replicate.
- Snapshot creation locks the associated tables until it is complete, so schedule your snapshot appropriately.
- Only push subscriptions are supported in Azure SQL Database. You can only add subscribers from the source database.

## Resolving database migration compatibility issues

There are a wide variety of compatibility issues that you might encounter, depending both on the version of SQL Server in the source database and the complexity of the database you are migrating. Older versions of SQL Server have more compatibility issues. Use the following resources, in addition to a targeted Internet search using your search engine of choice:

- [SQL Server database features not supported in Azure SQL Database](#)
- [Discontinued Database Engine Functionality in SQL Server 2016](#)
- [Discontinued Database Engine Functionality in SQL Server 2014](#)
- [Discontinued Database Engine Functionality in SQL Server 2012](#)
- [Discontinued Database Engine Functionality in SQL Server 2008 R2](#)
- [Discontinued Database Engine Functionality in SQL Server 2005](#)

In addition to searching the Internet and using these resources, use the [MSDN SQL Server community forums](#) or [StackOverflow](#).

**IMPORTANT**

SQL Database Managed Instance enables you to migrate an existing SQL Server instance and its databases with minimal to no compatibility issues. See [What is an Managed Instance](#).

## Next steps

- Use the script on the Azure SQL EMEA Engineers blog to [Monitor tempdb usage during migration](#).
- Use the script on the Azure SQL EMEA Engineers blog to [Monitor the transaction log space of your database while migration is occurring](#).
- For a SQL Server Customer Advisory Team blog about migrating using BACPAC files, see [Migrating from SQL Server to Azure SQL Database using BACPAC Files](#).
- For information about working with UTC time after migration, see [Modifying the default time zone for your local time zone](#).
- For information about changing the default language of a database after migration, see [How to change the default language of Azure SQL Database](#).

# New DBA in the cloud – Managing your single and pooled databases in Azure SQL Database

11/7/2019 • 28 minutes to read • [Edit Online](#)

Moving from the traditional self-managed, self-controlled environment to a PaaS environment can seem a bit overwhelming at first. As an app developer or a DBA, you would want to know the core capabilities of the platform that would help you keep your application available, performant, secure and resilient - always. This article aims to do exactly that. The article succinctly organizes resources and gives you some guidance on how to best use the key capabilities of SQL Database with single and pooled databases to manage and keep your application running efficiently and achieve optimal results in the cloud. Typical audience for this article would be those who:

- Are evaluating migration of their application(s) to Azure SQL Database – Modernizing your application(s).
- Are in the process of migrating their application(s) – On-going migration scenario.
- Have recently completed the migration to Azure SQL DB – New DBA in the cloud.

This article discusses some of the core characteristics of Azure SQL Database as a platform that you can readily leverage when working with single databases and pooled databases in elastic pools. They are the following:

- Monitor database using the Azure portal
- Business continuity and disaster recovery (BCDR)
- Security and compliance
- Intelligent database monitoring and maintenance
- Data movement

## NOTE

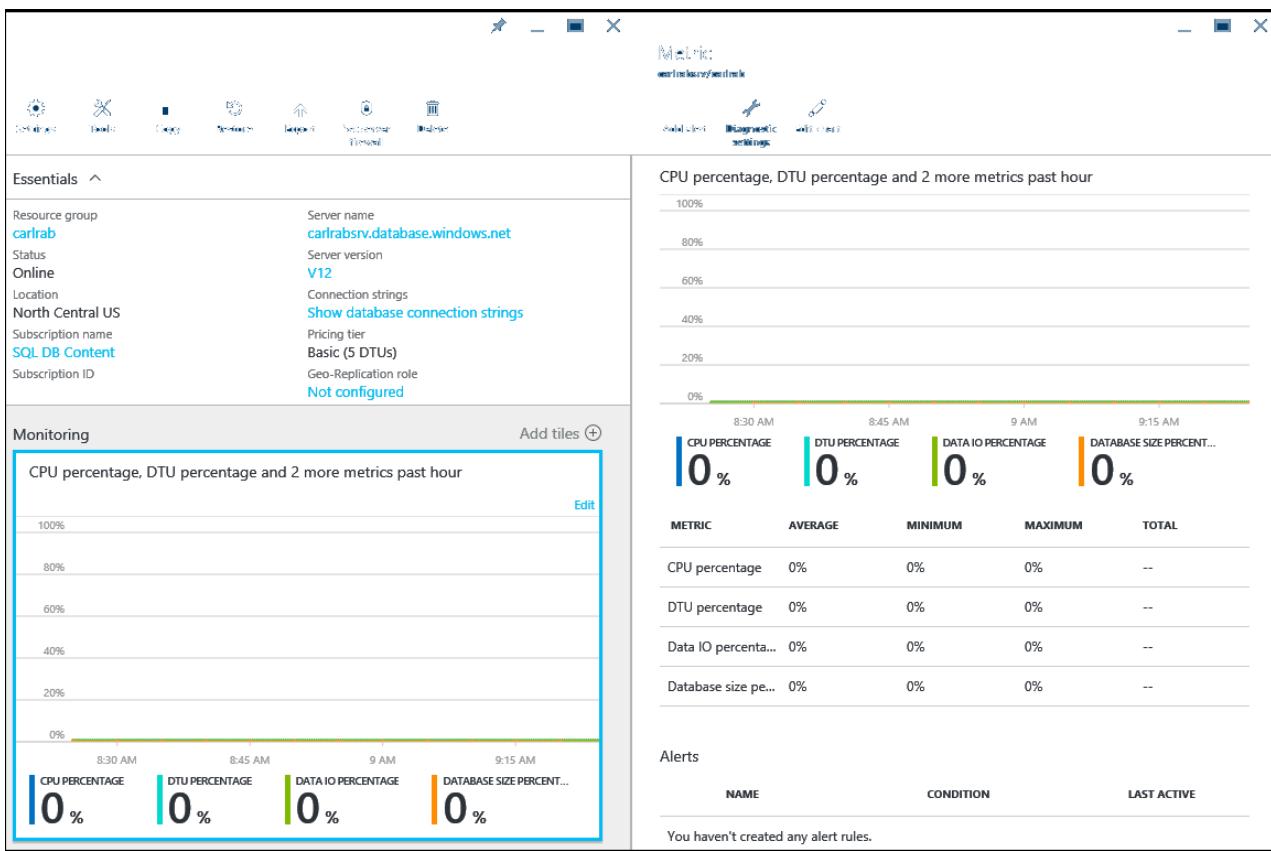
This article applies to the following deployment options in Azure SQL Database: single databases and elastic pools. It does not apply to the managed instance deployment option in SQL Database.

## Monitor databases using the Azure portal

In the [Azure portal](#), you can monitor an individual database utilization by selecting your database and clicking the **Monitoring** chart. This brings up a **Metric** window that you can change by clicking the **Edit chart** button. Add the following metrics:

- CPU percentage
- DTU percentage
- Data IO percentage
- Database size percentage

Once you've added these metrics, you can continue to view them in the **Monitoring** chart with more information on the **Metric** window. All four metrics show the average utilization percentage relative to the **DTU** of your database. See the [DTU-based purchasing model](#) and [vCore-based purchasing model](#) articles for more information about service tiers.



You can also configure alerts on the performance metrics. Click the **Add alert** button in the **Metric** window. Follow the wizard to configure your alert. You have the option to alert if the metrics exceed a certain threshold or if the metric falls below a certain threshold.

For example, if you expect the workload on your database to grow, you can choose to configure an email alert whenever your database reaches 80% on any of the performance metrics. You can use this as an early warning to figure out when you might have to switch to the next highest compute size.

The performance metrics can also help you determine if you are able to downgrade to a lower compute size. Assume you are using a Standard S2 database and all performance metrics show that the database on average does not use more than 10% at any given time. It is likely that the database will work well in Standard S1. However, be aware of workloads that spike or fluctuate before making the decision to move to a lower compute size.

## Business continuity and disaster recovery (BCDR)

Business continuity and disaster recovery abilities enable you to continue your business, as usual, in case of a disaster. The disaster could be a database level event (for example, someone mistakenly drops a crucial table) or a data-center level event (regional catastrophe, for example a tsunami).

### How do I create and manage backups on SQL Database

You don't create backups on Azure SQL DB and that is because you don't have to. SQL Database automatically backs up databases for you, so you no longer must worry about scheduling, taking and managing backups. The platform takes a full backup every week, differential backup every few hours and a log backup every 5 minutes to ensure the disaster recovery is efficient, and the data loss minimal. The first full backup happens as soon as you create a database. These backups are available to you for a certain period called the "Retention Period" and varies according to the service tier you choose. SQL Database provides you the ability to restore to any point in time within this retention period using [Point in Time Recovery \(PITR\)](#).

SERVICE TIER	RETENTION PERIOD IN DAYS
Basic	7
Standard	35
Premium	35

In addition, the [Long-Term Retention \(LTR\)](#) feature allows you to hold onto your backup files for a much longer period specifically, for up to 10 years, and restore data from these backups at any point within that period. Furthermore, the database backups are kept in geo-replicated storage to ensure resilience from regional catastrophe. You can also restore these backups in any Azure region at any point of time within the retention period. See [Business continuity overview](#).

### How do I ensure business continuity in the event of a datacenter-level disaster or regional catastrophe

Because your database backups are stored in geo-replicated storage to ensure that in case of a regional disaster, you can restore the backup to another Azure region. This is called geo-restore. The RPO (Recovery Point Objective) for this is generally < 1 hour and the ERT (Estimated Recovery Time – few minutes to hours).

For mission-critical databases, Azure SQL DB offers, active geo-replication. What this essentially does is that it creates a geo-replicated secondary copy of your original database in another region. For example, if your database is initially hosted in Azure West US region and you want regional disaster resilience. You'd create an active geo replica of the database in West US to say East US. When the calamity strikes on West US, you can fail over to the East US region. Configuring them in an auto-failover Group is even better because this ensures that the database automatically fails over to the secondary in East US in case of a disaster. The RPO for this is < 5 seconds and the ERT < 30 seconds.

If an auto-failover group is not configured, then your application needs to actively monitor for a disaster and initiate a failover to the secondary. You can create up to 4 such active geo-replicas in different Azure regions. It gets even better. You can also access these secondary active geo-replicas for read-only access. This comes in very handy to reduce latency for a geo-distributed application scenario.

### How does my disaster recovery plan change from on-premises to SQL Database

In summary, the traditional on-premises SQL Server setup required you to actively manage your Availability by using features such as Failover Clustering, Database Mirroring, Transaction Replication, or Log Shipping and maintain and manage backups to ensure Business Continuity. With SQL Database, the platform manages these for you, so you can focus on developing and optimizing your database application and not worry about disaster management as much. You can have backup and disaster recovery plans configured and working with just a few clicks on the Azure portal (or a few commands using the PowerShell APIs).

To learn more about Disaster recovery, see: [Azure SQL Db Disaster Recovery 101](#)

## Security and compliance

SQL Database takes Security and Privacy very seriously. Security within SQL Database is available at the database level and at the platform level and is best understood when categorized into several layers. At each layer you get to control and provide optimal security for your application. The layers are:

- Identity & authentication ([Windows/SQL authentication](#) and [Azure Active Directory \[AAD\] authentication](#)).
- Monitoring activity ([Auditing](#) and [threat detection](#)).
- Protecting actual data ([Transparent Data Encryption \[TDE\]](#) and [Always Encrypted \[AE\]](#)).
- Controlling Access to sensitive and privileged data ([Row Level security](#) and [Dynamic Data Masking](#)).

Azure Security Center offers centralized security management across workloads running in Azure, on-premises, and in other clouds. You can view whether essential SQL Database protection such as [Auditing](#) and [Transparent Data Encryption \[TDE\]](#) are configured on all resources, and create policies based on your own requirements.

## What user authentication methods are offered in SQL Database

There are [two authentication methods](#) offered in SQL Database:

- [Azure Active Directory Authentication](#)
- SQL authentication

The traditional windows authentication is not supported. Azure Active Directory (AD) is a centralized identity and access management service. With this you can very conveniently provide a Single Sign-on Access (SSO) to all the personnel in your organization. What this means is that the credentials are shared across all Azure services for simpler authentication. AAD supports [MFA \(Multi Factor Authentication\)](#) and with a [few clicks](#) AAD can be integrated with Windows Server Active Directory. SQL Authentication works exactly like you've been using it in the past. You provide a username/password and you can authenticate users to any database on a given SQL Database server. This also allows SQL Database and SQL Data Warehouse to offer multi-factor authentication and guest user accounts within an Azure AD domain. If you already have an Active Directory on-premises, you can federate the directory with Azure Active Directory to extend your directory to Azure.

IF YOU...	SQL DATABASE / SQL DATA WAREHOUSE
Prefer not to use Azure Active Directory (AD) in Azure	Use <a href="#">SQL authentication</a>
Used AD on SQL Server on-premises	<a href="#">Federate AD with Azure AD</a> , and use Azure AD authentication. With this, you can use Single Sign-On.
Need to enforce multi-factor authentication (MFA)	Require MFA as a policy through <a href="#">Microsoft Conditional Access</a> , and use <a href="#">Azure AD Universal authentication with MFA support</a> .
Have guest accounts from Microsoft accounts (live.com, outlook.com) or other domains (gmail.com)	Use <a href="#">Azure AD Universal authentication</a> in SQL Database/Data Warehouse, which leverages <a href="#">Azure AD B2B Collaboration</a> .
Are logged in to Windows using your Azure AD credentials from a federated domain	Use <a href="#">Azure AD integrated authentication</a> .
Are logged in to Windows using credentials from a domain not federated with Azure	Use <a href="#">Azure AD integrated authentication</a> .
Have middle-tier services which need to connect to SQL Database or SQL Data Warehouse	Use <a href="#">Azure AD integrated authentication</a> .

## How do I limit or control connectivity access to my database

There are multiple techniques at your disposal that you could use to attain optimal connectivity organization for your application.

- Firewall Rules
- VNet Service Endpoints
- Reserved IPs

### Firewall

A firewall prevents access to your server from an external entity by allowing only specific entities access to your SQL Database server. By default, all connections and databases inside the SQL Database server are disallowed,

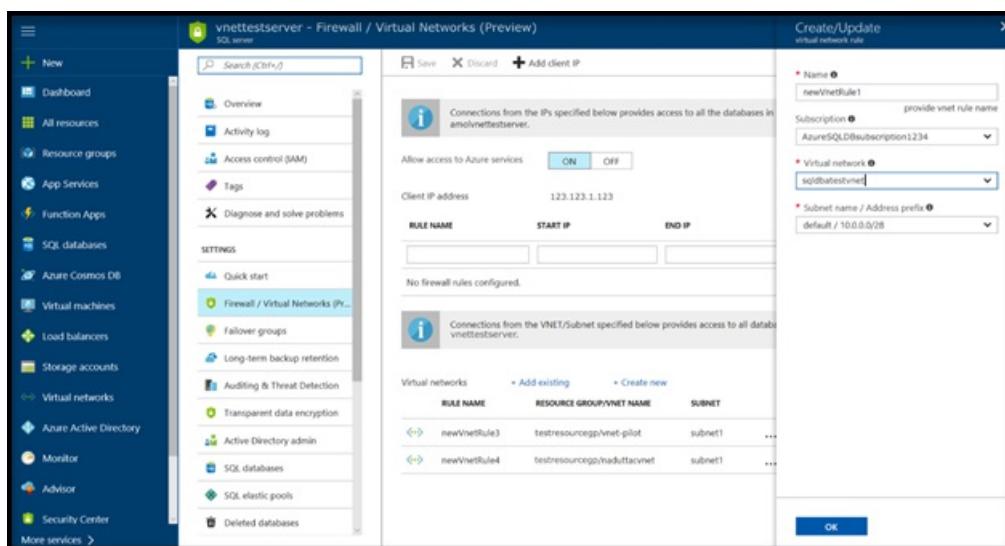
except connections coming in from other Azure Services. With a firewall rule you can open access to your server only to entities (for example, a developer machine) that you approve of, by allowing that computer's IP address through the firewall. It also allows you to specify a range of IPs that you would want to allow access to the SQL Database server. For example, developer machine IP addresses in your organization can be added at once by specifying a range in the Firewall settings page.

You can create firewall rules at the server level or at the database level. Server level IP firewall rules can either be created using the Azure portal or with SSMS. For learning more about how to set a server-level and database-level firewall rule, see: [Create IP firewall rules in SQL Database](#).

#### Service endpoints

By default, your SQL database is configured to "Allow Azure services to access server" – which means any Virtual Machine in Azure may attempt to connect to your database. These attempts still do have to get authenticated. However, if you would not like your database to be accessible by any Azure IPs, you can disable "Allow Azure services to access server". Additionally, you can configure [VNet Service Endpoints](#).

Service endpoints (SE) allow you to expose your critical Azure resources only to your own private virtual network in Azure. By doing so, you essentially eliminate public access to your resources. The traffic between your virtual network to Azure stays on the Azure backbone network. Without SE you get forced-tunneling packet routing. Your virtual network forces the internet traffic to your organization and the Azure Service traffic to go over the same route. With Service Endpoints, you can optimize this since the packets flow straight from your virtual network to the service on Azure backbone network.



#### Reserved IPs

Another option is to provision [reserved IPs](#) for your VMs, and add those specific VM IP addresses in the server firewall settings. By assigning reserved IPs, you save the trouble of having to update the firewall rules with changing IP addresses.

#### What port do I connect to SQL Database on

Port 1433. SQL Database communicates over this port. To connect from within a corporate network, you have to add an outbound rule in the firewall settings of your organization. As a guideline, avoid exposing port 1433 outside the Azure boundary.

#### How can I monitor and regulate activity on my server and database in SQL Database

##### SQL Database Auditing

With SQL Database, you can turn ON Auditing to track database events. [SQL Database Auditing](#) records database events and writes them into an audit log file in your Azure Storage Account. Auditing is especially useful if you intend to gain insight into potential security and policy violations, maintain regulatory compliance etc. It allows you to define and configure certain categories of events that you think need auditing and based on that you can get preconfigured reports and a dashboard to get an overview of events occurring on your database. You can

apply these auditing policies either at the database level or at the server level. A guide on how to turn on auditing for your server/database, see: [Enable SQL Database Auditing](#).

#### Threat detection

With [threat detection](#), you get the ability to act upon security or policy violations discovered by Auditing very easily. You don't need to be a security expert to address potential threats or violations in your system. Threat detection also has some built-in capabilities like SQL Injection detection. SQL Injection is an attempt to alter or compromise the data and a quite common way of attacking a database application in general. Threat detection runs multiple sets of algorithms which detect potential vulnerabilities and SQL injection attacks, as well as anomalous database access patterns (such as access from an unusual location or by an unfamiliar principal). Security officers or other designated administrators receive an email notification if a threat is detected on the database. Each notification provides details of the suspicious activity and recommendations on how to further investigate and mitigate the threat. To learn how to turn on Threat detection, see: [Enable threat detection](#).

#### How do I protect my data in general on SQL Database

Encryption provides a strong mechanism to protect and secure your sensitive data from intruders. Your encrypted data is of no use to the intruder without the decryption key. Thus, it adds an extra layer of protection on top of the existing layers of security built in SQL Database. There are two aspects to protecting your data in SQL Database:

- Your data that is at-rest in the data and log files
- Your data that is in-flight

In SQL Database, by default, your data at rest in the data and log files on the storage subsystem is completely and always encrypted via [Transparent Data Encryption \[TDE\]](#). Your backups are also encrypted. With TDE there are no changes required on your application side that is accessing this data. The encryption and decryption happen transparently; hence the name. For protecting your sensitive data in-flight and at rest, SQL Database provides a feature called [Always Encrypted \(AE\)](#). AE is a form of client-side encryption which encrypts sensitive columns in your database (so they are in ciphertext to database administrators and unauthorized users). The server receives the encrypted data to begin with. The key for Always Encrypted is also stored on the client side, so only authorized clients can decrypt the sensitive columns. The server and data administrators cannot see the sensitive data since the encryption keys are stored on the client. AE encrypts sensitive columns in the table end to end, from unauthorized clients to the physical disk. AE supports equality comparisons today, so DBAs can continue to query encrypted columns as part of their SQL commands. Always Encrypted can be used with a variety of key store options, such as [Azure Key Vault](#), Windows certificate store, and local hardware security modules.

CHARACTERISTICS	ALWAYS ENCRYPTED	TRANSPARENT DATA ENCRYPTION
<b>Encryption span</b>	End-to-end	At-rest data
<b>Database server can access sensitive data</b>	No	Yes, since encryption is for the data at rest
<b>Allowed T-SQL operations</b>	Equality comparison	All T-SQL surface area is available
<b>App changes required to use the feature</b>	Minimal	Very Minimal
<b>Encryption granularity</b>	Column level	Database level

#### How can I limit access to sensitive data in my database

Every application has a certain bit of sensitive data in the database that needs to be protected from being visible to everyone. Certain personnel within the organization need to view this data, however others shouldn't be able to view this data. One example is employee wages. A manager would need access to the wage information for their

direct reports however, the individual team members shouldn't have access to the wage information of their peers. Another scenario is data developers who might be interacting with sensitive data during development stages or testing, for example, SSNs of customers. This information again doesn't need to be exposed to the developer. In such cases, your sensitive data either needs to be masked or not be exposed at all. SQL Database offers two such approaches to prevent unauthorized users from being able to view sensitive data:

[Dynamic Data Masking](#) is a data masking feature that enables you to limit sensitive data exposure by masking it to non-privileged users on the application layer. You define a masking rule that can create a masking pattern (for example, to only show last four digits of a national ID SSN: XXX-XX-0000 and mark most of it as Xs) and identify which users are to be excluded from the masking rule. The masking happens on-the-fly and there are various masking functions available for various data categories. Dynamic data masking allows you to automatically detect sensitive data in your database and apply masking to it.

[Row Level security](#) enables you to control access at the row level. Meaning, certain rows in a database table based on the user executing the query (group membership or execution context) are hidden. The access restriction is done on the database tier instead of in an application tier, to simplify your app logic. You start by creating a filter predicate, filtering out rows that are not be exposed and the security policy next defining who has access to these rows. Finally, the end user runs their query and, depending on the user's privilege, they either view those restricted rows or are unable to see them at all.

## How do I manage encryption keys in the cloud

There are key management options for both Always Encrypted (client-side encryption) and Transparent Data Encryption (encryption at rest). It's recommended that you regularly rotate encryption keys. The rotation frequency should align with both your internal organization regulations and compliance requirements.

### Transparent Data Encryption (TDE)

There is a two-key hierarchy in TDE – the data in each user database is encrypted by a symmetric AES-256 database-unique database encryption key (DEK), which in turn is encrypted by a server-unique asymmetric RSA 2048 master key. The master key can be managed either:

- Automatically by the platform - SQL Database.
- Or by you using [Azure Key Vault](#) as the key store.

By default, the master key for Transparent Data Encryption is managed by the SQL Database service for convenience. If your organization would like control over the master key, there is an option to use Azure Key Vault] (sql-database-always-encrypted-azure-key-vault.md) as the key store. By using Azure Key Vault, your organization assumes control over key provisioning, rotation, and permission controls. [Rotation or switching the type of a TDE master key](#) is fast, as it only re-encrypts the DEK. For organizations with separation of roles between security and data management, a security admin could provision the key material for the TDE master key in Azure Key Vault and provide an Azure Key Vault key identifier to the database administrator to use for encryption at rest on a server. The Key Vault is designed such that Microsoft does not see or extract any encryption keys. You also get a centralized management of keys for your organization.

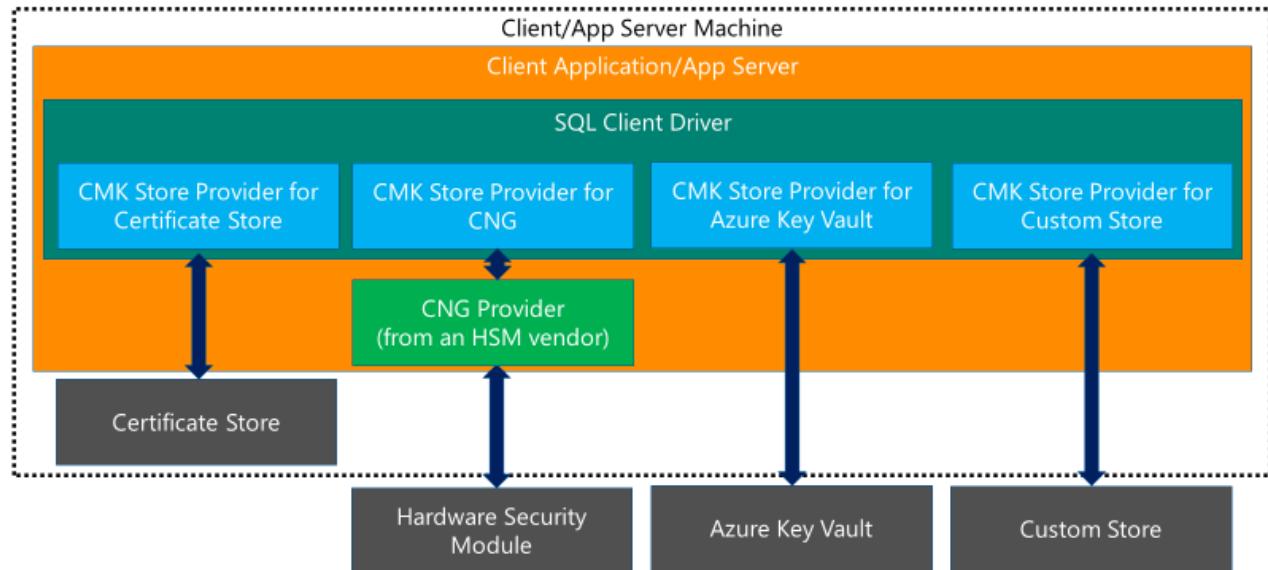
### Always Encrypted

There is also a [two-key hierarchy](#) in Always Encrypted - a column of sensitive data is encrypted by an AES 256-column encryption key (CEK), which in turn is encrypted by a column master key (CMK). The client drivers provided for Always Encrypted have no limitations on the length of CMKs. The encrypted value of the CEK is stored on the database, and the CMK is stored in a trusted key store, such as Windows Certificate Store, Azure Key Vault, or a hardware security module.

- Both the [CEK and CMK](#) can be rotated.
- CEK rotation is a size of data operation and can be time-intensive depending on the size of the tables containing the encrypted columns. Hence it is prudent to plan CEK rotations accordingly.
- CMK rotation, however, does not interfere with database performance, and can be done with separated roles.

The following diagram shows the key store options for the column master keys in Always Encrypted

# Always Encrypted CMK Store Providers



## How can I optimize and secure the traffic between my organization and SQL Database

The network traffic between your organization and SQL Database would generally get routed over the public network. However, if you choose to optimize this path and make it more secure, you can look into Express Route. Express route essentially lets you extend your corporate network into the Azure platform over a private connection. By doing so, you do not go over the public Internet. You also get higher security, reliability, and routing optimization that translates to lower network latencies and much faster speeds than you would normally experience going over the public internet. If you are planning on transferring a significant chunk of data between your organization and Azure, using Express Route can yield cost benefits. You can choose from three different connectivity models for the connection from your organization to Azure:

- [Cloud Exchange Co-location](#)
- [Any-to-any](#)
- [Point-to-Point](#)

Express Route also allows you to burst up to 2x the bandwidth limit you purchase for no additional charge. It is also possible to configure cross region connectivity using Express route. To see a list of ER connectivity providers, see: [Express Route Partners and Peering Locations](#). The following articles describe Express Route in more detail:

- [Introduction on Express Route](#)
- [Prerequisites](#)
- [Workflows](#)

## Is SQL Database compliant with any regulatory requirements, and how does that help with my own organization's compliance

SQL Database is compliant with a range of regulatory compliances. To view the latest set of compliances that have been met by SQL Database, visit the [Microsoft Trust Center](#) and drill down on the compliances that are important to your organization to see if SQL Database is included under the compliant Azure services. It is important to note that although SQL Database may be certified as a compliant service, it aids in the compliance of your organization's service but does not automatically guarantee it.

## Intelligent database monitoring and maintenance after migration

Once you've migrated your database to SQL Database, you are going to want to monitor your database (for

example, check how the resource utilization is like or DBCC checks) and perform regular maintenance (for example, rebuild or reorganize indexes, statistics etc.). Fortunately, SQL Database is Intelligent in the sense that it uses the historical trends and recorded metrics and statistics to proactively help you monitor and maintain your database, so that your application runs optimally always. In some cases, Azure SQL DB can automatically perform maintenance tasks depending on your configuration setup. There are three facets to monitoring your database in SQL Database:

- Performance monitoring and optimization.
- Security optimization.
- Cost optimization.

### Performance monitoring and optimization

With Query Performance Insights, you can get tailored recommendations for your database workload so that your applications can keep running at an optimal level - always. You can also set it up so that these recommendations get applied automatically and you do not have to bother performing maintenance tasks. With Index Advisor, you can automatically implement index recommendations based on your workload - this is called Auto-Tuning. The recommendations evolve as your application workload changes to provide you with the most relevant suggestions. You also get the option to manually review these recommendations and apply them at your discretion.

### Security optimization

SQL Database provides actionable security recommendations to help you secure your data and threat detection for identifying and investigating suspicious database activities that may pose a potential threat to the database.

[Vulnerability assessment](#) is a database scanning and reporting service that allows you to monitor the security state of your databases at scale and identify security risks and drift from a security baseline defined by you. After every scan, a customized list of actionable steps and remediation scripts is provided, as well as an assessment report that can be used to help meet compliance requirements.

With Azure Security Center, you identify the security recommendations across the board and apply them with a single click.

### Cost optimization

Azure SQL platform analyzes the utilization history across the databases in a server to evaluate and recommend cost-optimization options for you. This analysis usually takes a fortnight to analyze and build up actionable recommendations. Elastic pool is one such option. The recommendation appears on the portal as a banner:

DATABASE	STATUS	PRICING TIER
BeijingDB	Online	Standard: S3
BuenosAiresDB	Online	Standard: S3

You can also view this analysis under the "Advisor" section:

The screenshot shows the 'Advisor recommendations' section of the Azure portal. It displays 2 of 7 selected subscriptions. The 'Cost (1)' tab is selected, showing 1 total recommendation. The recommendation details are: Impact: High (1), Description: 'Use SQL elastic database pools', and Potential MoM: 3.

## How do I monitor the performance and resource utilization in SQL Database

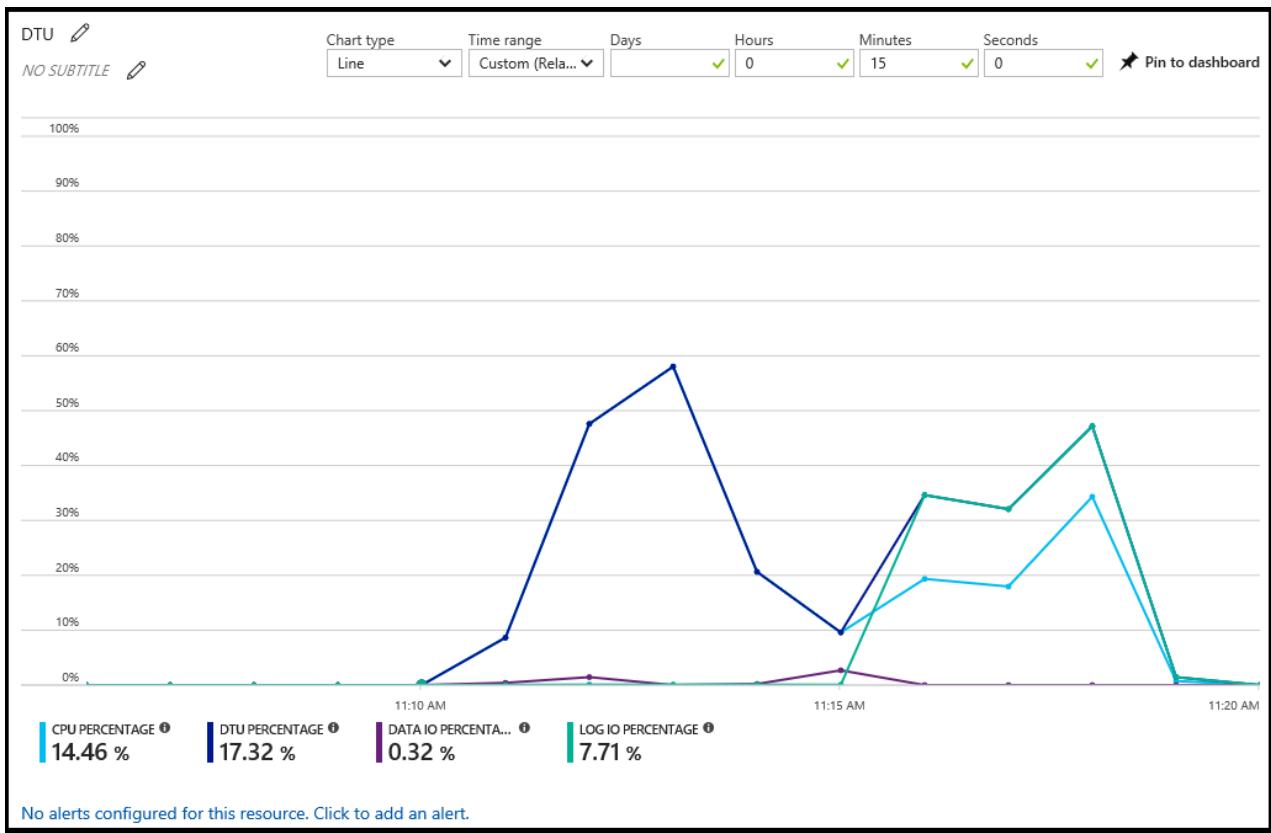
In SQL Database you can leverage the intelligent insights of the platform to monitor the performance and tune accordingly. You can monitor performance and resource utilization in SQL Database using the following methods:

### Azure portal

The Azure portal shows a database's utilization by selecting the database and clicking the chart in the Overview pane. You can modify the chart to show multiple metrics, including CPU percentage, DTU percentage, Data IO percentage, Sessions percentage, and Database size percentage.

The screenshot shows the Azure portal monitoring dashboard for a database named 'carlrb'. The left sidebar shows 'Essentials' and 'Monitoring'. The 'Monitoring' section is highlighted with a blue border and displays four metrics: CPU percentage, DTU percentage, Data IO percentage, and Database size percentage, all currently at 0%.

Metric	Average	Minimum	Maximum	Total
CPU percentage	0%	0%	0%	--
DTU percentage	0%	0%	0%	--
Data IO percentage	0%	0%	0%	--
Database size percentage	0%	0%	0%	--



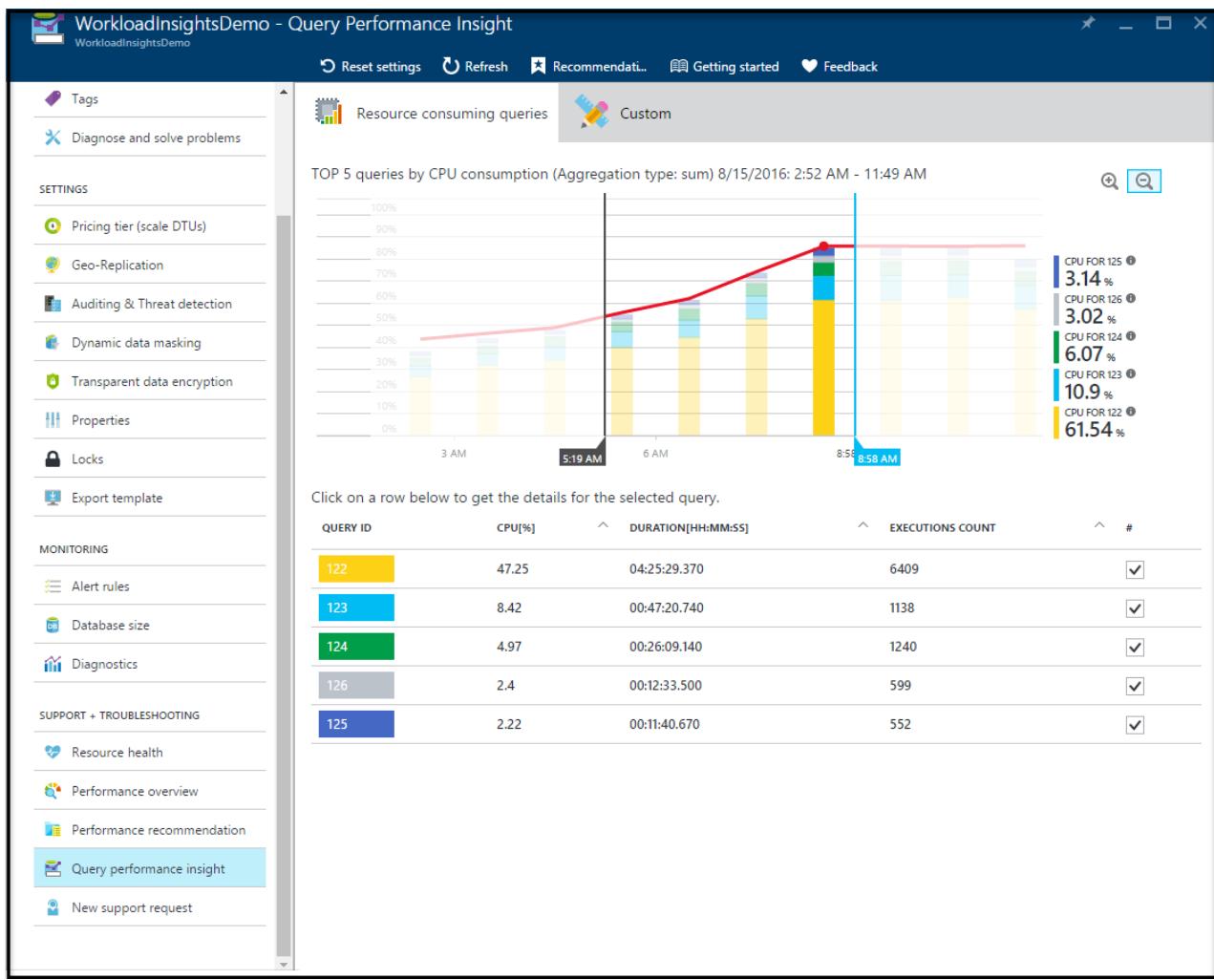
From this chart, you can also configure alerts by resource. These alerts allow you to respond to resource conditions with an email, write to an HTTPS/HTTP endpoint or perform an action. For more information, see [Create alerts](#).

#### Dynamic Management Views

You can query the `sys.dm_db_resource_stats` dynamic management view to return resource consumption statistics history from the last hour and the `sys.resource_stats` system catalog view to return history for the last 14 days.

#### Query Performance Insight

[Query Performance Insight](#) allows you to see a history of the top resource-consuming queries and long-running queries for a specific database. You can quickly identify TOP queries by resource utilization, duration, and frequency of execution. You can track queries and detect regression. This feature requires [Query Store](#) to be enabled and active for the database.



#### Azure SQL Analytics (Preview) in Azure Monitor logs

[Azure Monitor logs](#) allows you to collect and visualize key Azure SQL database performance metrics, supporting up to 150,000 SQL Databases and 5,000 SQL Elastic pools per workspace. You can use it to monitor and receive notifications. You can monitor SQL Database and elastic pool metrics across multiple Azure subscriptions and elastic pools and can be used to identify issues at each layer of an application stack.

#### I am noticing performance issues: How does my SQL Database troubleshooting methodology differ from SQL Server

A major portion of the troubleshooting techniques you would use for diagnosing query and database performance issues remain the same. After all the same SQL Server engine powers the cloud. However, the platform - Azure SQL DB has built in 'intelligence'. It can help you troubleshoot and diagnose performance issues even more easily. It can also perform some of these corrective actions on your behalf and in some cases, proactively fix them - automatically.

Your approach towards troubleshooting performance issues can significantly benefit by using intelligent features such as [Query Performance Insight\(QPI\)](#) and [Database Advisor](#) in conjunction and so the difference in methodology differs in that respect – you no longer need to do the manual work of grinding out the essential details that might help you troubleshoot the issue at hand. The platform does the hard work for you. One example of that is QPI. With QPI, you can drill all the way down to the query level and look at the historical trends and figure out when exactly the query regressed. The Database Advisor gives you recommendations on things that might help you improve your overall performance in general like - missing indexes, dropping indexes, parameterizing your queries etc.

With performance troubleshooting, it is important to identify whether it is just the application or the database backing it, that's impacting your application performance. Often the performance problem lies in the application layer. It could be the architecture or the data access pattern. For example, consider you have a chatty application that is sensitive to network latency. In this case, your application suffers because there would be many short

requests going back and forth ("chatty") between the application and the server and on a congested network, these roundtrips add up fast. To improve the performance in this case, you can use [Batch Queries](#). Using batches helps you tremendously because now your requests get processed in a batch; thus, helping you cut down on the roundtrip latency and improve your application performance.

Additionally, if you notice a degradation in the overall performance of your database, you can monitor the [sys.dm\\_db\\_resource\\_stats](#) and [sys.resource\\_stats](#) dynamic management views in order to understand CPU, IO, and memory consumption. Your performance maybe impacted because your database is starved of resources. It could be that you may need to change the compute size and/or service tier based on the growing and shrinking workload demands.

For a comprehensive set of recommendations for tuning performance issues, see: [Tune your database](#).

### How do I ensure I am using the appropriate service tier and compute size

SQL Database offers various service tiers Basic, Standard, and Premium. Each service tier you get a guaranteed predictable performance tied to that service tier. Depending on your workload, you may have bursts of activity where your resource utilization might hit the ceiling of the current compute size that you are in. In such cases, it is useful to first start by evaluating whether any tuning can help (for example, adding or altering an index etc.). If you still encounter limit issues, consider moving to a higher service tier or compute size.

SERVICE TIER	COMMON USE CASE SCENARIOS
<b>Basic</b>	Applications with a handful users and a database that doesn't have high concurrency, scale, and performance requirements.
<b>Standard</b>	Applications with a considerable concurrency, scale, and performance requirements coupled with low to medium IO demands.
<b>Premium</b>	Applications with lots of concurrent users, high CPU/memory, and high IO demands. High concurrency, high throughput, and latency sensitive apps can leverage the Premium level.

For making sure you're on the right compute size, you can monitor your query and database resource consumption through one of the above-mentioned ways in "How do I monitor the performance and resource utilization in SQL Database". Should you find that your queries/databases are consistently running hot on CPU/Memory etc. you can consider scaling up to a higher compute size. Similarly, if you note that even during your peak hours, you don't seem to use the resources as much; consider scaling down from the current compute size.

If you have a SaaS app pattern or a database consolidation scenario, consider using an Elastic pool for cost optimization. Elastic pool is a great way to achieve database consolidation and cost-optimization. To read more about managing multiple databases using elastic pool, see: [Manage pools and databases](#).

### How often do I need to run database integrity checks for my database

SQL Database uses some smart techniques that allow it to handle certain classes of data corruption automatically and without any data loss. These techniques are built in to the service and are leveraged by the service when need arises. On a regular basis, your database backups across the service are tested by restoring them and running DBCC CHECKDB on it. If there are issues, SQL Database proactively addresses them. [Automatic page repair](#) is leveraged for fixing pages that are corrupt or have data integrity issues. The database pages are always verified with the default CHECKSUM setting that verifies the integrity of the page. SQL Database proactively monitors and reviews the data integrity of your database and, if issues arise, addresses them with the highest priority. In addition to these, you may choose to optionally run your own integrity checks at your will. For more information, see [Data Integrity in SQL Database](#)

# Data movement after migration

## How do I export and import data as BACPAC files from SQL Database

- **Export:** You can export your Azure SQL database as a BACPAC file from the Azure portal

The screenshot shows the Azure portal interface for managing a database. On the left, there's a sidebar with various icons. In the main area, a database named 'mysampleDB' is selected. The 'Overview' tab is currently active, indicated by a red box. At the top right of the main content area, there are several buttons: 'Copy', 'Restore', 'Export' (which is highlighted with a red box), 'Set server firewall', 'Delete', 'Connect with...', and 'Feedback'. Below these buttons, detailed information about the database is listed, including its resource group ('mysampleRG'), status ('Online'), location ('West US 2'), subscription details, and connection strings.

- **Import:** You can also import data as a BACPAC file into the database using the Azure portal.

This screenshot shows the Azure portal interface for managing a SQL server. A SQL server named 'mysamplesrvr' is selected, with its 'Overview' tab active (highlighted with a red box). At the top right, there are buttons for 'New database', 'New pool', 'New data warehouse', and 'Import database' (which is highlighted with a red box). The main pane displays the server's configuration, including its resource group ('mysampleRG'), status ('Available'), location ('West US 2'), and server admin ('carlrb').

## How do I synchronize data between SQL Database and SQL Server

You have several ways to achieve this:

- **Data Sync** – This feature helps you synchronize data bi-directionally between multiple on-premises SQL Server databases and SQL Database. To sync with on-premises SQL Server databases, you need to install and configure sync agent on a local computer and open the outbound TCP port 1433.
- **Transaction Replication** – With transaction replication you can synchronize your data from on-premises to Azure SQL DB with the on-premises being the publisher and the Azure SQL DB being the subscriber. For now, only this setup is supported. For more information on how to migrate your data from on-premises to Azure SQL with minimal downtime, see: [Use Transaction Replication](#)

## Next steps

Learn about [SQL Database](#).

# Replication to SQL Database single and pooled databases

12/10/2019 • 3 minutes to read • [Edit Online](#)

SQL Server replication can be configured to single and pooled databases on a [SQL Database server](#) in Azure SQL Database.

## Supported Configurations:

- The SQL Server can be an instance of SQL Server running on-premises or an instance of SQL Server running in an Azure virtual machine in the cloud. For more information, see [SQL Server on Azure Virtual Machines overview](#).
- The Azure SQL database must be a push subscriber of a SQL Server publisher.
- The distribution database and the replication agents cannot be placed on an Azure SQL database.
- Snapshot and one-way transactional replication are supported. Peer-to-peer transactional replication and merge replication are not supported.
- Replication is available for public preview on Azure SQL Database Managed Instance. Managed Instance can host publisher, distributor, and subscriber databases. For more information, see [Replication with SQL Database Managed Instance](#).

## Versions

On-premises SQL Server publishers and distributors must be using (at least) one of the following versions:

- SQL Server 2016 and greater
- SQL Server 2014 [RTM CU10 \(12.0.4427.24\)](#) or [SP1 CU3 \(12.0.2556.4\)](#)
- SQL Server 2012 [SP2 CU8 \(11.0.5634.1\)](#) or [SP3 \(11.0.6020.0\)](#)

### NOTE

Attempting to configure replication using an unsupported version can result in error number MSSQL\_REPL20084 (The process could not connect to Subscriber.) and MSSQL\_REPL40532 (Cannot open server <name> requested by the login. The login failed.).

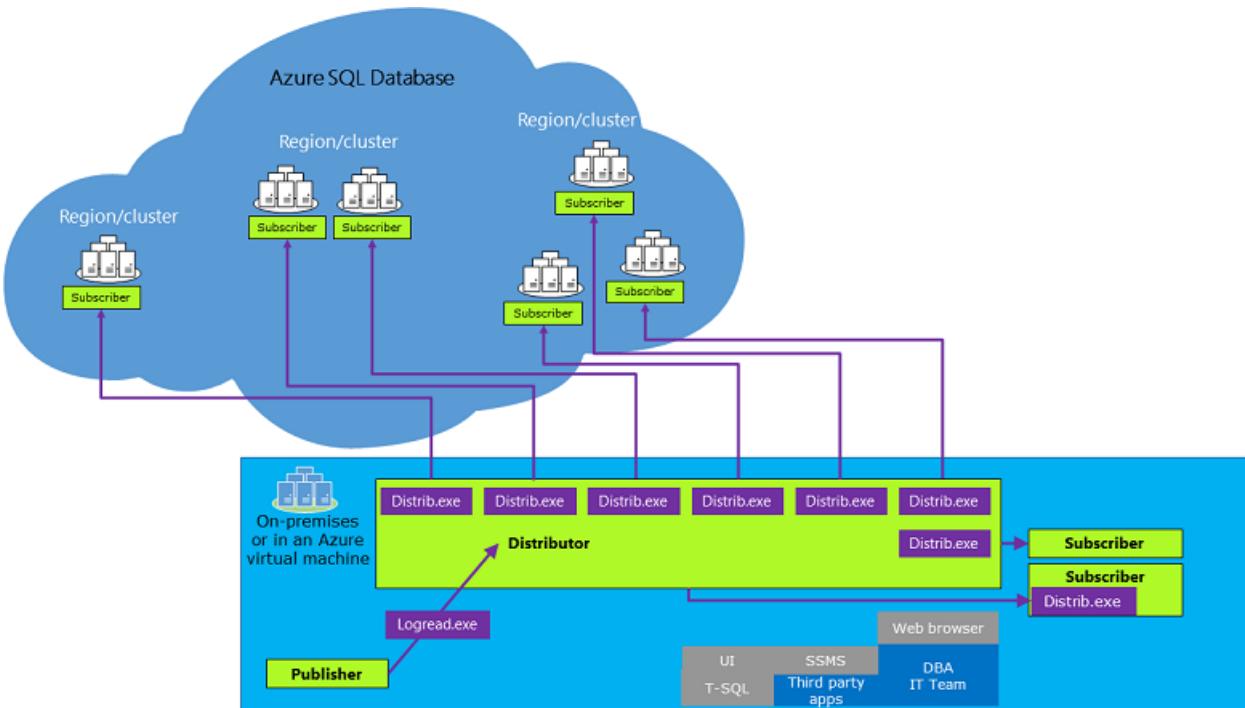
To use all the features of Azure SQL Database, you must be using the latest versions of [SQL Server Management Studio](#) and [SQL Server Data Tools](#).

## Remarks

- Replication can be configured by using [SQL Server Management Studio](#) or by executing Transact-SQL statements on the publisher. You cannot configure replication by using the Azure portal.
- Replication can only use SQL Server authentication logins to connect to an Azure SQL database.
- Replicated tables must have a primary key.
- You must have an existing Azure subscription.
- The Azure SQL database subscriber can be in any region.
- A single publication on SQL Server can support both Azure SQL Database and SQL Server (on-premises and SQL Server in an Azure virtual machine) subscribers.

- Replication management, monitoring, and troubleshooting must be performed from the on-premises SQL Server.
- Only push subscriptions to Azure SQL Database are supported.
- Only `@subscriber_type = 0` is supported in **sp\_addsubscription** for SQL Database.
- Azure SQL Database does not support bi-directional, immediate, updatable, or peer to peer replication.

## Replication Architecture



## Scenarios

### Typical Replication Scenario

1. Create a transactional replication publication on an on-premises SQL Server database.
2. On the on-premises SQL Server use the **New Subscription Wizard** or Transact-SQL statements to create a push to subscription to Azure SQL Database.
3. With single and pooled databases in Azure SQL Database, the initial data set is a snapshot that is created by the Snapshot Agent and distributed and applied by the Distribution Agent. With a managed instance database, you can also use a database backup to seed the subscriber database.

### Data Migration Scenario

1. Use transactional replication to replicate data from an on-premises SQL Server database to Azure SQL Database.
2. Redirect the client or middle-tier applications to update the Azure SQL database copy.
3. Stop updating the SQL Server version of the table and remove the publication.

## Limitations

The following options are not supported for Azure SQL Database subscriptions:

- Copy file groups association
- Copy table partitioning schemes
- Copy index partitioning schemes
- Copy user defined statistics
- Copy default bindings

- Copy rule bindings
- Copy fulltext indexes
- Copy XML XSD
- Copy XML indexes
- Copy permissions
- Copy spatial indexes
- Copy filtered indexes
- Copy data compression attribute
- Copy sparse column attribute
- Convert filestream to MAX data types
- Convert hierarchyid to MAX data types
- Convert spatial to MAX data types
- Copy extended properties
- Copy permissions

#### **Limitations to be determined**

- Copy collation
- Execution in a serialized transaction of the SP

## Examples

Create a publication and a push subscription. For more information, see:

- [Create a Publication](#)
- [Create a Push Subscription](#) by using the Azure SQL Database server name as the subscriber (for example `N'azuresqldbdns.database.windows.net'`) and the Azure SQL database name as the destination database (for example **AdventureWorks**).

## See Also

- [Transactional replication](#)
- [Create a Publication](#)
- [Create a Push Subscription](#)
- [Types of Replication](#)
- [Monitoring \(Replication\)](#)
- [Initialize a Subscription](#)

# Azure SQL Database Advanced Threat Protection for single or pooled databases

11/7/2019 • 2 minutes to read • [Edit Online](#)

Advanced Threat Protection for single and pooled databases detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases. Advanced Threat Protection can identify **Potential SQL injection**, **Access from unusual location or data center**, **Access from unfamiliar principal or potentially harmful application**, and **Brute force SQL credentials** - see more details in [Advanced Threat Protection alerts](#).

You can receive notifications about the detected threats via [email notifications](#) or [Azure portal](#)

Advanced Threat Protection is part of the [advanced data security](#) (ADS) offering, which is a unified package for advanced SQL security capabilities. Advanced Threat Protection can be accessed and managed via the central SQL ADS portal.

## Set up Advanced Threat Protection in the Azure portal

1. Launch the Azure portal at <https://portal.azure.com>.
2. Navigate to the configuration page of the Azure SQL Database server you want to protect. In the security settings, select **Advanced Data Security**.
3. On the **Advanced Data Security** configuration page:
  - Enable Advanced Data Security on the server.
  - In **Advanced Threat Protection Settings**, in the **Send alerts to** text box, provide the list of emails to receive security alerts upon detection of anomalous database activities.

The screenshot shows the Azure portal interface for managing a SQL server named 'vanazuresqldbserver'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with sub-links like Quick start, Failover groups, Manage Backups, Active Directory admin, SQL databases, SQL elastic pools, Deleted databases, Import/Export history, DTU quota, Properties, Locks, Export template), and Security. The 'Advanced Data Security' link is highlighted with a red box. The main content area is titled 'ADVANCED DATA SECURITY' with a status switch between 'ON' and 'OFF'. A note states that Advanced Data Security costs 15 USD/server/month and includes Data Discovery & Classification, Vulnerability Assessment, and Advanced Threat Protection. Below this is a 'VULNERABILITY ASSESSMENT SETTINGS' section with links for Subscription (SQL DB Content) and Storage account. Under 'Periodic recurring scans', the status is set to 'OFF'. There are fields for 'Send scan reports to' and a checkbox for 'Also send email notification to admins and subscription owners'. The 'ADVANCED THREAT PROTECTION SETTINGS' section is also highlighted with a red box. It includes a 'Send alerts to' field containing 'Email addresses' with a green checkmark, and a checked checkbox for 'Also send email notification to admins and subscription owners'. Other settings include 'Advanced Threat Protection types' (set to 'All').

#### NOTE

Prices in screenshots does not always reflect the current price, and are an example.

## Set up Advanced Threat Protection using PowerShell

For a script example, see [Configure auditing and Advanced Threat Protection using PowerShell](#).

## Next steps

- Learn more about [Advanced Threat Protection](#).
- Learn more about [Advanced Threat Protection in managed instance](#).
- Learn more about [advanced data security](#).
- Learn more about [auditing](#)
- Learn more about [Azure security center](#)
- For more information on pricing, see the [SQL Database pricing page](#)

# Manage Azure SQL Database long-term backup retention

11/22/2019 • 6 minutes to read • [Edit Online](#)

In Azure SQL Database, you can configure a single or a pooled database with a [long-term backup retention](#) policy (LTR) to automatically retain the database backups in separate Azure Blob storage containers for up to 10 years. You can then recover a database using these backups using the Azure portal or PowerShell.

## IMPORTANT

Azure SQL database managed instance does not currently support long-term backup retention.

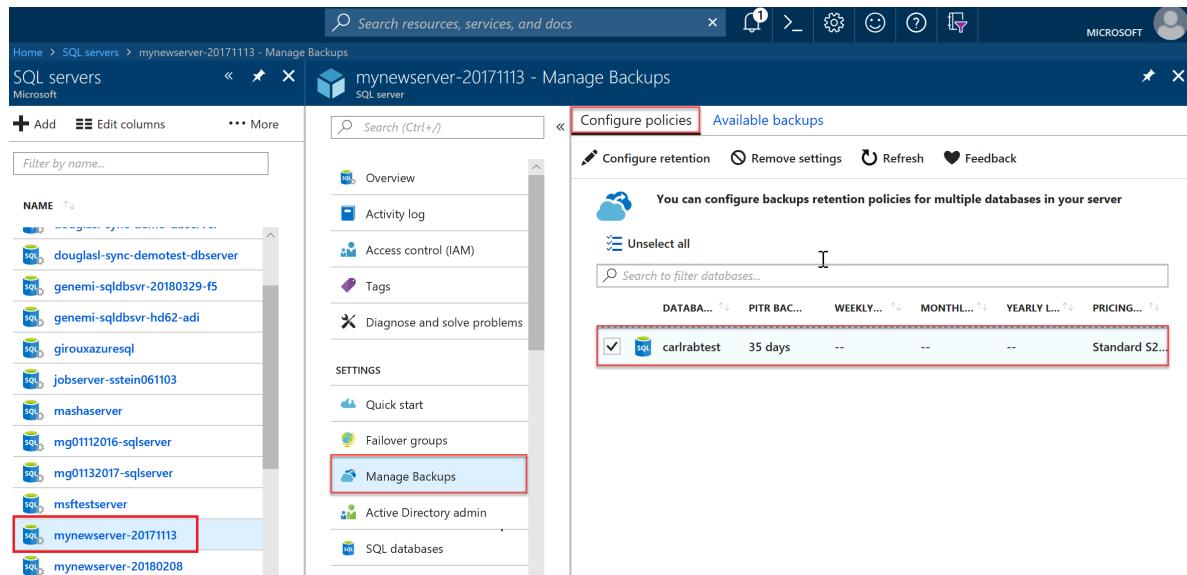
## Using Azure portal

The following sections show you how to use the Azure portal to configure the long-term retention, view backups in long-term retention, and restore backup from long-term retention.

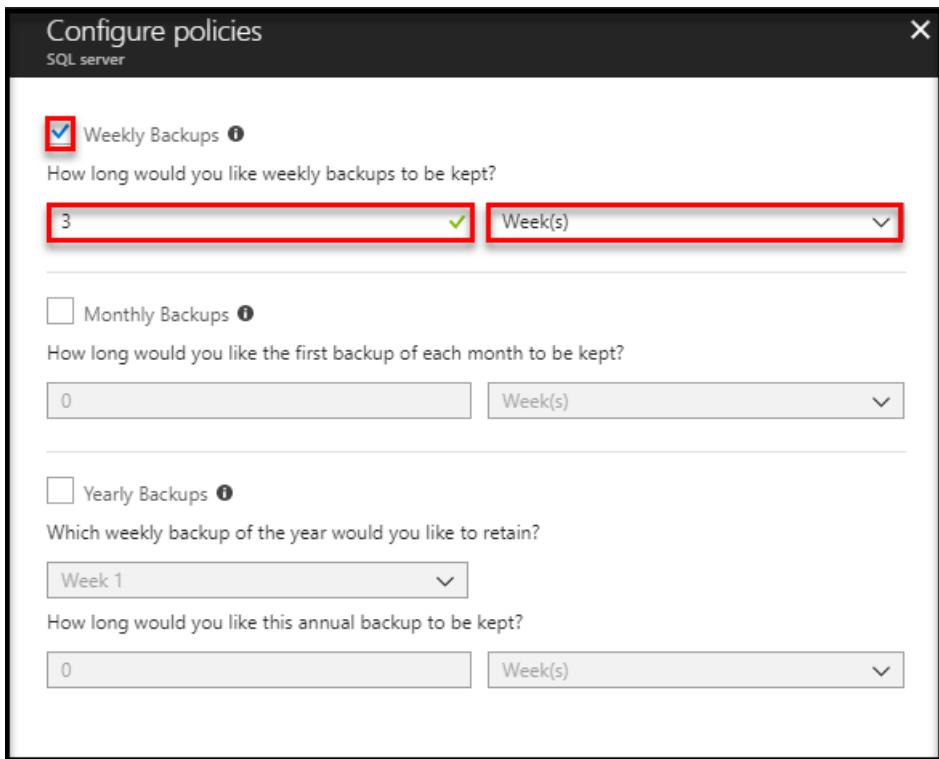
### Configure long-term retention policies

You can configure SQL Database to [retain automated backups](#) for a period longer than the retention period for your service tier.

1. In the Azure portal, select your SQL server and then click **Manage Backups**. On the **Configure policies** tab, select the checkbox for the database on which you want to set or modify long-term backup retention policies. If the checkbox next to the database is not selected, the changes for the policy will not apply to that database.



2. In the **Configure policies** pane, select if want to retain weekly, monthly or yearly backups and specify the retention period for each.



3. When complete, click **Apply**.

#### IMPORTANT

When you enable a long-term backup retention policy, it may take up to 7 days for the first backup to become visible and available to restore. For details of the LTR backup cadence, see [long-term backup retention](#).

### View backups and restore from a backup

View the backups that are retained for a specific database with a LTR policy, and restore from those backups.

1. In the Azure portal, select your SQL server and then click **Manage Backups**. On the **Available backups** tab, select the database for which you want to see available backups.

The screenshot shows the 'Available backups' pane in the Azure portal. It lists databases with available backups, including 'testdb\_2018-03-31T10-40Z', 'testdb', 'testdb\_2018-03-26T13-08Z\_newsku', and 'testdb\_2018-03-26T13-08Z\_pool2'. The 'testdb' database is selected and highlighted with a red box.

DATABASE	DATABASE DROPPED TIME	SERVER NAME
testdb_2018-03-31T10-40Z	--	trgrie-ltr-server
testdb	--	trgrie-ltr-server
testdb_2018-03-26T13-08Z_newsku	--	trgrie-ltr-server
testdb_2018-03-26T13-08Z_pool2	--	trgrie-ltr-server

2. In the **Available backups** pane, review the available backups.

A screenshot of the Azure portal interface. On the left, there's a sidebar with a 'Delete' button. The main area shows a table titled 'Databases with available backups'. A search bar at the top says 'Search to filter backups...'. The table has two columns: 'BACKUP TIME' and 'BACKUP EXPIRATION TIME'. A specific row is highlighted with a red box, showing '2018-03-26 06:08' in the first column and '2018-04-02 06:08' in the second.

3. Select the backup from which you want to restore, and then specify the new database name.

A screenshot of the 'testdb' Long-term retention restore dialog. It includes fields for 'Database name' (set to 'testdb\_ltr'), 'Select a backup' (set to '2018-03-26 06:08'), and 'Target server' ('trgrie-ltr-server northeurope'). The 'Database name' field and the 'Select a backup' dropdown are both highlighted with red boxes.

4. Click **OK** to restore your database from the backup in Azure SQL storage to the new database.

5. On the toolbar, click the notification icon to view the status of the restore job.

A screenshot of the 'Sources' notification center. It displays two items: 'Restoring SQL database with long ...' and 'Deployment started...'. The notification bell icon on the toolbar is highlighted with a red box.

6. When the restore job is completed, open the **SQL databases** page to view the newly restored database.

#### NOTE

From here, you can connect to the restored database using SQL Server Management Studio to perform needed tasks, such as to extract a bit of data from the restored database to copy into the existing database or to delete the existing database and rename the restored database to the existing database name.

# Using PowerShell

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

The following sections show you how to use PowerShell to configure the long-term backup retention, view backups in Azure SQL storage, and restore from a backup in Azure SQL storage.

## RBAC roles to manage long-term retention

For **Get-AzSqlDatabaseLongTermRetentionBackup** and **Restore-AzSqlDatabase**, you will need to have one of the following roles:

- Subscription Owner role or
- SQL Server Contributor role or
- Custom role with the following permissions:

Microsoft.Sql/locations/longTermRetentionBackups/read

Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionBackups/read

Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/read

For **Remove-AzSqlDatabaseLongTermRetentionBackup**, you will need to have one of the following roles:

- Subscription Owner role or
- Custom role with the following permission:

Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/delete

## NOTE

The SQL Server Contributor role does not have permission to delete LTR backups.

RBAC permissions could be granted in either *subscription* or *resource group* scope. However, to access LTR backups that belong to a dropped server, the permission must be granted in the *subscription* scope of that server.

- Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/delete

## Create an LTR policy

```

get the SQL server
$subId = "<subscriptionId>"
$serverName = "<serverName>"
$resourceGroup = "<resourceGroupName>"
$dbName = "<databaseName>"

Connect-AzAccount
Select-AzSubscription -SubscriptionId $subId

$server = Get-AzSqlServer -ServerName $serverName -ResourceGroupName $resourceGroup

create LTR policy with WeeklyRetention = 12 weeks. MonthlyRetention and YearlyRetention = 0 by default.
Set-AzSqlDatabaseBackupLongTermRetentionPolicy -ServerName $serverName -DatabaseName $dbName `
-ResourceGroupName $resourceGroup -WeeklyRetention P12W

create LTR policy with WeeklyRetention = 12 weeks, YearlyRetention = 5 years and WeekOfYear = 16 (week of
April 15). MonthlyRetention = 0 by default.
Set-AzSqlDatabaseBackupLongTermRetentionPolicy -ServerName $serverName -DatabaseName $dbName `
-ResourceGroupName $resourceGroup -WeeklyRetention P12W -YearlyRetention P5Y -WeekOfYear 16

```

## View LTR policies

This example shows how to list the LTR policies within a server

```

get all LTR policies within a server
$ltrPolicies = Get-AzSqlDatabase -ResourceGroupName Default-SQL-WestCentralUS -ServerName trgrive-ltr-server |

Get-AzSqlDatabaseLongTermRetentionPolicy -Current

get the LTR policy of a specific database
$ltrPolicies = Get-AzSqlDatabaseBackupLongTermRetentionPolicy -ServerName $serverName -DatabaseName $dbName `
-ResourceGroupName $resourceGroup -Current

```

## Clear an LTR policy

This example shows how to clear an LTR policy from a database

```

Set-AzSqlDatabaseBackupLongTermRetentionPolicy -ServerName $serverName -DatabaseName $dbName `
-ResourceGroupName $resourceGroup -RemovePolicy

```

## View LTR backups

This example shows how to list the LTR backups within a server.

```

get the list of all LTR backups in a specific Azure region
backups are grouped by the logical database id, within each group they are ordered by the timestamp, the
earliest backup first
$ltrBackups = Get-AzSqlDatabaseLongTermRetentionBackup -Location $server.Location

get the list of LTR backups from the Azure region under the named server
$ltrBackups = Get-AzSqlDatabaseLongTermRetentionBackup -Location $server.Location -ServerName $serverName

get the LTR backups for a specific database from the Azure region under the named server
$ltrBackups = Get-AzSqlDatabaseLongTermRetentionBackup -Location $server.Location -ServerName $serverName -
DatabaseName $dbName

list LTR backups only from live databases (you have option to choose All/Live/Deleted)
$ltrBackups = Get-AzSqlDatabaseLongTermRetentionBackup -Location $server.Location -DatabaseState Live

only list the latest LTR backup for each database
$ltrBackups = Get-AzSqlDatabaseLongTermRetentionBackup -Location $server.Location -ServerName $serverName -
OnlyLatestPerDatabase

```

## Delete LTR backups

This example shows how to delete an LTR backup from the list of backups.

```
remove the earliest backup
$ltrBackup = $ltrBackups[0]
Remove-AzSqlDatabaseLongTermRetentionBackup -ResourceId $ltrBackup.ResourceId
```

### IMPORTANT

Deleting LTR backup is non-reversible. To delete an LTR backup after the server has been deleted you must have Subscription scope permission. You can set up notifications about each delete in Azure Monitor by filtering for operation 'Deletes a long term retention backup'. The activity log contains information on who and when made the request. See [Create activity log alerts](#) for detailed instructions.

## Restore from LTR backups

This example shows how to restore from an LTR backup. Note, this interface did not change but the resource id parameter now requires the LTR backup resource id.

```
restore a specific LTR backup as an P1 database on the server $serverName of the resource group
$resourceGroup
Restore-AzSqlDatabase -FromLongTermRetentionBackup -ResourceId $ltrBackup.ResourceId -ServerName $serverName
-ResourceGroupName $resourceGroup `
 -TargetDatabaseName $dbName -ServiceObjectiveName P1
```

### IMPORTANT

To restore from an LTR backup after the server has been deleted, you must have permissions scoped to the server's subscription and that subscription must be active. You must also omit the optional -ResourceGroupName parameter.

### NOTE

From here, you can connect to the restored database using SQL Server Management Studio to perform needed tasks, such as to extract a bit of data from the restored database to copy into the existing database or to delete the existing database and rename the restored database to the existing database name. See [point in time restore](#).

## Next steps

- To learn about service-generated automatic backups, see [automatic backups](#)
- To learn about long-term backup retention, see [long-term backup retention](#)

# Create, configure, and manage elastic jobs

2/11/2020 • 3 minutes to read • [Edit Online](#)

In this article, you will learn how to create, configure, and manage elastic jobs.

If you have not used Elastic jobs, [learn more about the job automation concepts in Azure SQL Database](#).

## Create and configure the agent

1. Create or identify an empty S0 or higher SQL database. This database will be used as the *Job database* during Elastic Job agent creation.
2. Create an Elastic Job agent in the [portal](#) or with [PowerShell](#).

The screenshot shows the Azure portal interface for creating an Elastic Job agent. On the left, the 'Elastic Job agent' blade is open, displaying the following information:

- An icon of a blue arrow pointing down next to a clock, with the text: "An Elastic Job agent runs jobs whose definitions are stored in an Azure SQL Database. A job is a T-SQL script that is scheduled or executed ad-hoc against a group of Azure SQL databases."
- A link labeled "Learn more".
- Fields for "Name" (set to "ConstosoAgent") and "Subscription" (set to "ContosoSubscription").
- A section titled "Preview terms" with "Accepted" checked.
- A section titled "Job database" with a link to "Configure required settings".
- A checkbox for "Pin to dashboard".
- Buttons for "Create" and "Automation options".

On the right, the "Job database" blade is open, showing the following details:

- A dropdown menu for "Select server" set to "contososerver".
- A search bar with placeholder text "Search to filter databases...".
- A table with columns "DATABASE NAME" and "PRICING TIER". It lists one database entry: "JobDatabase" under "Standard S0: 10 D...".
- A large blue info icon with the text: "An Elastic Job agent database must have a service level objective of S0 or above. Databases with lower service level objectives are not shown."
- A button labeled "OK".

## Create, run, and manage jobs

1. Create a credential for job execution in the *Job database* using [PowerShell](#) or [T-SQL](#).
2. Define the target group (the databases you want to run the job against) using [PowerShell](#) or [T-SQL](#).
3. Create a job agent credential in each database the job will run ([add the user \(or role\) to each database in the group](#)). For an example, see the [PowerShell tutorial](#).
4. Create a job using [PowerShell](#) or [T-SQL](#).

5. Add job steps using [PowerShell](#) or [T-SQL](#).
6. Run a job using [PowerShell](#) or [T-SQL](#).
7. Monitor job execution status using the portal, [PowerShell](#) or [T-SQL](#).

The screenshot shows the Azure portal interface for managing an Elastic Job agent named 'contosoagent'. The 'Overview' tab is selected. On the left, there's a sidebar with various navigation links. The main content area displays basic resource group details and a table of recent job execution logs.

STATUS	JOB NAME	JOB EXECUTION ID	START TIME	END TIME	DURATION
In Progress	Job1	05594811-cc30-4c38-bd0-d5deac7960c	6/12/2018, 10:48:02 PM		
Succeeded	Job1	89cb0588-f89a-4e4e-a25e-3628d58d3da9	6/12/2018, 10:46:21 PM	6/12/2018, 10:46:30 PM	10s
Succeeded	Job1	c89c507c-433f-464c-856f-c6edb9d51ba7	6/12/2018, 10:45:59 PM	6/12/2018, 10:46:16 PM	16s
Succeeded	Job1	9241889c-3dae-4374-bf25-57640cf5ab0e	6/12/2018, 8:29:20 PM	6/12/2018, 8:29:32 PM	12s

## Credentials for running jobs

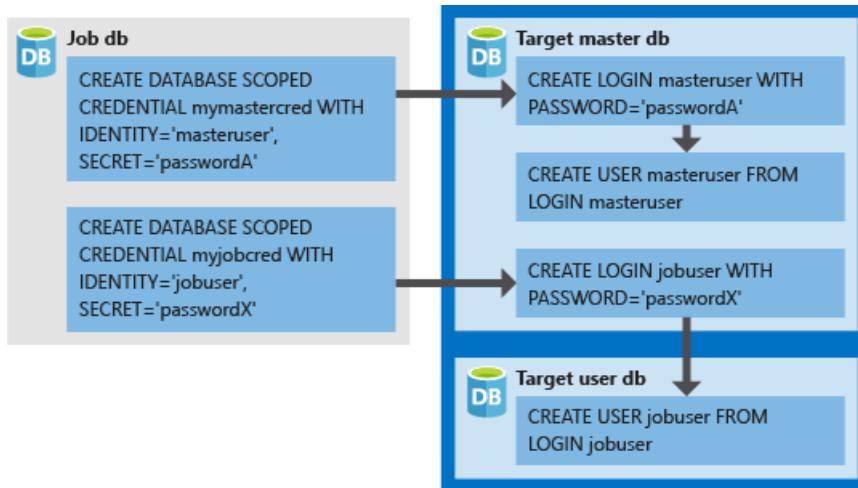
Jobs use [database scoped credentials](#) to connect to the databases specified by the target group upon execution. If a target group contains servers or pools, these database scoped credentials are used to connect to the master database to enumerate the available databases.

Setting up the proper credentials to run a job can be a little confusing, so keep the following points in mind:

- The database scoped credentials must be created in the *Job database*.
- **All target databases must have a login with sufficient permissions for the job to complete successfully** (`jobuser` in the diagram below).
- Credentials can be reused across jobs, and the credential passwords are encrypted and secured from users who have read-only access to job objects.

The following image is designed to assist in understanding and setting up the proper job credentials.

**Remember to create the user in every database (all target user dbs) the job needs to run.**



## Security best practices

A few best practice considerations for working with Elastic Jobs:

- Limit usage of the APIs to trusted individuals.

- Credentials should have the least privileges necessary to perform the job step. For more information, see [Authorization and Permissions SQL Server](#).
- When using a server and/or pool target group member, it is highly suggested to create a separate credential with rights on the master database to view/list databases that is used to expand the database lists of the server(s) and/or pool(s) prior to the job execution.

## Agent performance, capacity, and limitations

Elastic Jobs use minimal compute resources while waiting for long-running jobs to complete.

Depending on the size of the target group of databases and the desired execution time for a job (number of concurrent workers), the agent requires different amounts of compute and performance of the *Job database* (the more targets and the higher number of jobs, the higher the amount of compute required).

Currently, the preview is limited to 100 concurrent jobs.

### Prevent jobs from reducing target database performance

To ensure resources aren't overburdened when running jobs against databases in a SQL elastic pool, jobs can be configured to limit the number of databases a job can run against at the same time.

Set the number of concurrent databases a job runs on by setting the `sp_add_jobstep` stored procedure's `@max_parallelism` parameter in T-SQL, or `Add-AzSqlElasticJobStep -MaxParallelism` in PowerShell.

## Best practices for creating jobs

### Idempotent scripts

A job's T-SQL scripts must be **idempotent**. **Idempotent** means that if the script succeeds, and it is run again, the same result occurs. A script may fail due to transient network issues. In that case, the job will automatically retry running the script a preset number of times before desisting. An idempotent script has the same result even if its been successfully run twice (or more).

A simple tactic is to test for the existence of an object before creating it.

```
IF NOT EXISTS (some_object)
 -- Create the object
 -- If it exists, drop the object before recreating it.
```

Similarly, a script must be able to execute successfully by logically testing for and countering any conditions it finds.

## Next steps

- [Create and manage Elastic Jobs using PowerShell](#)
- [Create and manage Elastic Jobs using Transact-SQL \(T-SQL\)](#)

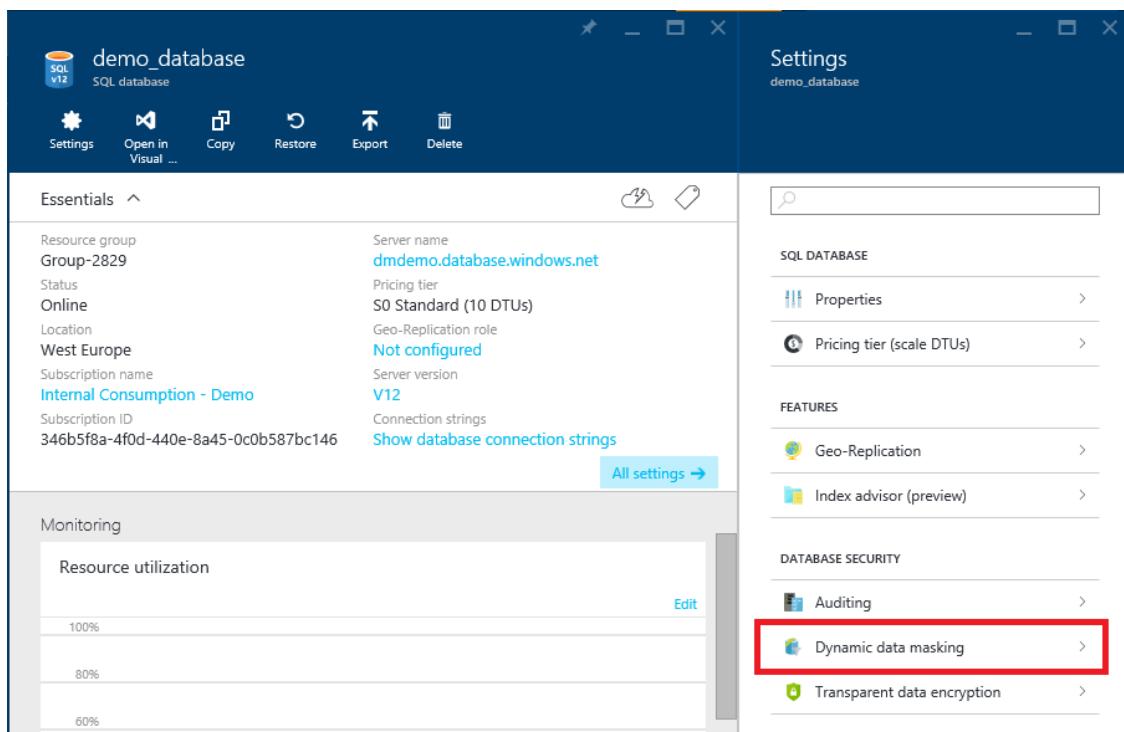
# Get started with SQL Database dynamic data masking with the Azure portal

1/23/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to implement [dynamic data masking](#) with the Azure portal. You can also implement dynamic data masking using [Azure SQL Database cmdlets](#) or the [REST API](#).

## Set up dynamic data masking for your database using the Azure portal

1. Launch the Azure portal at <https://portal.azure.com>.
2. Navigate to the settings page of the database that includes the sensitive data you want to mask.
3. Click the **Dynamic Data Masking** tile that launches the **Dynamic Data Masking** configuration page.
  - Alternatively, you can scroll down to the **Operations** section and click **Dynamic Data Masking**.



4. In the **Dynamic Data Masking** configuration page, you may see some database columns that the recommendations engine has flagged for masking. In order to accept the recommendations, just click **Add Mask** for one or more columns and a mask is created based on the default type for this column. You can change the masking function by clicking on the masking rule and editing the masking field format to a different format of your choice. Be sure to click **Save** to save your settings.

The screenshot shows the 'Dynamic Data Masking' configuration page for the 'demo\_database'. At the top, there are 'Save', 'Discard', and 'Add Mask' buttons. A message box at the top right says 'Downlevel clients require the use of Security Enabled Connection Strings.' Below this, the 'Masking Rules' section is empty, stating 'You haven't created any masking rules.' A note below it says 'SQL users excluded from masking (administrators are always excluded)' with a checked checkbox. A red box highlights the 'Recommended fields to mask' section, which lists columns from the 'Customer' and 'CustomerAddress' tables in the 'SalesLT' schema that are recommended for masking. Each row has an 'ADD MASK' button.

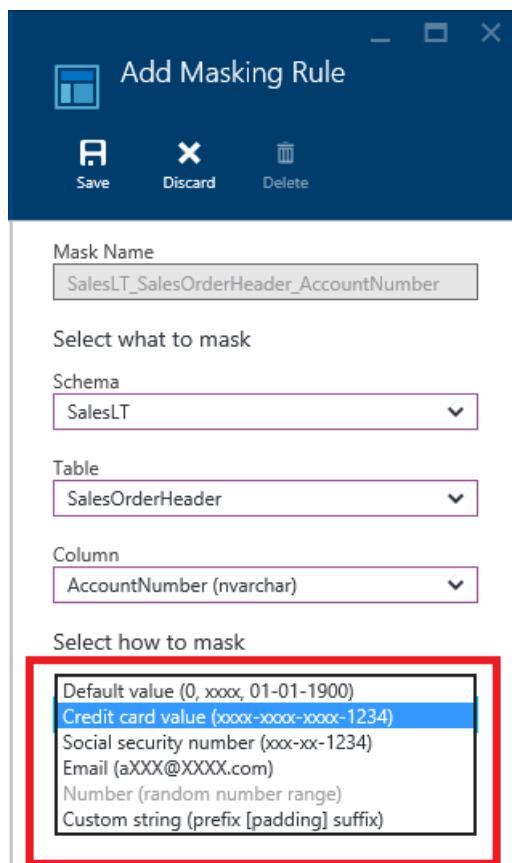
SCHEMA	TABLE	COLUMN	
SalesLT	Customer	FirstName	<b>ADD MASK</b>
SalesLT	Customer	LastName	<b>ADD MASK</b>
SalesLT	Customer	EmailAddress	<b>ADD MASK</b>
SalesLT	Customer	Phone	<b>ADD MASK</b>
SalesLT	CustomerAddress	AddressID	<b>ADD MASK</b>

5. To add a mask for any column in your database, at the top of the **Dynamic Data Masking** configuration page, click **Add Mask** to open the **Add Masking Rule** configuration page.

The screenshot shows the 'Add Masking Rule' configuration page. The 'Add Mask' button in the top bar is highlighted with a red box. The 'Masking Rules' section is empty. A note below it says 'SQL users excluded from masking (administrators are always excluded)' with a checked checkbox.

6. Select the **Schema**, **Table** and **Column** to define the designated field for masking.

7. Choose a **Masking Field Format** from the list of sensitive data masking categories.



8. Click **Save** in the data masking rule page to update the set of masking rules in the dynamic data masking policy.

9. Type the SQL users or AAD identities that should be excluded from masking, and have access to the unmasked sensitive data. This should be a semicolon-separated list of users. Users with administrator privileges always have access to the original unmasked data.

SQL users excluded from masking (administrators are always excluded) ⓘ

SQL users excluded from masking (administrators are always excluded)



#### TIP

To make it so the application layer can display sensitive data for application privileged users, add the SQL user or AAD identity the application uses to query the database. It is highly recommended that this list contain a minimal number of privileged users to minimize exposure of the sensitive data.

10. Click **Save** in the data masking configuration page to save the new or updated masking policy.

## Next steps

- For an overview of dynamic data masking, see [dynamic data masking](#).
- You can also implement dynamic data masking using [Azure SQL Database cmdlets](#) or the [REST API](#).

2 minutes to read

# Configure and manage Azure SQL Database security for geo-restore or failover

11/7/2019 • 4 minutes to read • [Edit Online](#)

This article describes the authentication requirements to configure and control [active geo-replication](#) and [auto-failover groups](#). It also provides the steps required to set up user access to the secondary database. Finally, it also describes how to enable access to the recovered database after using [geo-restore](#). For more information on recovery options, see [Business Continuity Overview](#).

## Disaster recovery with contained users

Unlike traditional users, which must be mapped to logins in the master database, a contained user is managed completely by the database itself. This has two benefits. In the disaster recovery scenario, the users can continue to connect to the new primary database or the database recovered using geo-restore without any additional configuration, because the database manages the users. There are also potential scalability and performance benefits from this configuration from a login perspective. For more information, see [Contained Database Users - Making Your Database Portable](#).

The main trade-off is that managing the disaster recovery process at scale is more challenging. When you have multiple databases that use the same login, maintaining the credentials using contained users in multiple databases may negate the benefits of contained users. For example, the password rotation policy requires that changes be made consistently in multiple databases rather than changing the password for the login once in the master database. For this reason, if you have multiple databases that use the same user name and password, using contained users is not recommended.

## How to configure logins and users

If you are using logins and users (rather than contained users), you must take extra steps to ensure that the same logins exist in the master database. The following sections outline the steps involved and additional considerations.

### NOTE

It is also possible to use Azure Active Directory (AAD) logins to manage your databases. For more information, see [Azure SQL logins and users](#).

### Set up user access to a secondary or recovered database

In order for the secondary database to be usable as a read-only secondary database, and to ensure proper access to the new primary database or the database recovered using geo-restore, the master database of the target server must have the appropriate security configuration in place before the recovery.

The specific permissions for each step are described later in this topic.

Preparing user access to a geo-replication secondary should be performed as part configuring geo-replication. Preparing user access to the geo-restored databases should be performed at any time when the original server is online (e.g. as part of the DR drill).

#### NOTE

If you fail over or geo-restore to a server that does not have properly configured logins, access to it will be limited to the server admin account.

Setting up logins on the target server involves three steps outlined below:

#### 1. Determine logins with access to the primary database

The first step of the process is to determine which logins must be duplicated on the target server. This is accomplished with a pair of SELECT statements, one in the logical master database on the source server and one in the primary database itself.

Only the server admin or a member of the **LoginManager** server role can determine the logins on the source server with the following SELECT statement.

```
SELECT [name], [sid]
FROM [sys].[sql_logins]
WHERE [type_desc] = 'SQL_Login'
```

Only a member of the db\_owner database role, the dbo user, or server admin, can determine all of the database user principals in the primary database.

```
SELECT [name], [sid]
FROM [sys].[database_principals]
WHERE [type_desc] = 'SQL_USER'
```

#### 2. Find the SID for the logins identified in step 1

By comparing the output of the queries from the previous section and matching the SIDs, you can map the server login to database user. Logins that have a database user with a matching SID have user access to that database as that database user principal.

The following query can be used to see all of the user principals and their SIDs in a database. Only a member of the db\_owner database role or server admin can run this query.

```
SELECT [name], [sid]
FROM [sys].[database_principals]
WHERE [type_desc] = 'SQL_USER'
```

#### NOTE

The **INFORMATION\_SCHEMA** and **sys** users have *NULL* SIDs, and the **guest** SID is **0x00**. The **dbo** SID may start with **0x010600000000164800000000048454**, if the database creator was the server admin instead of a member of **DbManager**.

#### 3. Create the logins on the target server

The last step is to go to the target server, or servers, and generate the logins with the appropriate SIDs. The basic syntax is as follows.

```
CREATE LOGIN [<login name>]
WITH PASSWORD = <login password>,
SID = <desired login SID>
```

#### NOTE

If you want to grant user access to the secondary, but not to the primary, you can do that by altering the user login on the primary server by using the following syntax.

```
ALTER LOGIN <login name> DISABLE
```

DISABLE doesn't change the password, so you can always enable it if needed.

## Next steps

- For more information on managing database access and logins, see [SQL Database security: Manage database access and login security](#).
- For more information on contained database users, see [Contained Database Users - Making Your Database Portable](#).
- To learn about active geo-replication, see [Active geo-replication](#).
- To learn about auto-failover groups, see [Auto-failover groups](#).
- For information about using geo-restore, see [geo-restore](#)

# In-Memory sample

11/7/2019 • 9 minutes to read • [Edit Online](#)

In-Memory technologies in Azure SQL Database enable you to improve performance of your application, and potentially reduce cost of your database. By using In-Memory technologies in Azure SQL Database, you can achieve performance improvements with various workloads.

In this article you'll see two samples that illustrate the use of In-Memory OLTP, as well as columnstore indexes in Azure SQL Database.

For more information, see:

- [In-Memory OLTP Overview and Usage Scenarios](#) (includes references to customer case studies and information to get started)
- [Documentation for In-Memory OLTP](#)
- [Columnstore Indexes Guide](#)
- Hybrid transactional/analytical processing (HTAP), also known as [real-time operational analytics](#)

## 1. Install the In-Memory OLTP sample

You can create the AdventureWorksLT sample database with a few clicks in the [Azure portal](#). Then, the steps in this section explain how you can enrich your AdventureWorksLT database with In-Memory OLTP objects and demonstrate performance benefits.

For a more simplistic, but more visually appealing performance demo for In-Memory OLTP, see:

- Release: [in-memory-oltp-demo-v1.0](#)
- Source code: [in-memory-oltp-demo-source-code](#)

### Installation steps

1. In the [Azure portal](#), create a Premium or Business Critical database on a server. Set the **Source** to the AdventureWorksLT sample database. For detailed instructions, see [Create your first Azure SQL database](#).
2. Connect to the database with SQL Server Management Studio ([SSMS.exe](#)).
3. Copy the [In-Memory OLTP Transact-SQL script](#) to your clipboard. The T-SQL script creates the necessary In-Memory objects in the AdventureWorksLT sample database that you created in step 1.
4. Paste the T-SQL script into SSMS, and then execute the script. The `MEMORY_OPTIMIZED = ON` clause CREATE TABLE statements are crucial. For example:

```
CREATE TABLE [SalesLT].[SalesOrderHeader_inmem] (
 [SalesOrderID] int IDENTITY NOT NULL PRIMARY KEY NONCLUSTERED ...,
 ...
) WITH (MEMORY_OPTIMIZED = ON);
```

### Error 40536

If you get error 40536 when you run the T-SQL script, run the following T-SQL script to verify whether the database supports In-Memory:

```
SELECT DatabasePropertyEx(DB_Name(), 'IsXTPSupported');
```

A result of **0** means that In-Memory isn't supported, and **1** means that it is supported. To diagnose the problem, ensure that the database is at the Premium service tier.

#### About the created memory-optimized items

**Tables:** The sample contains the following memory-optimized tables:

- SalesLT.Product\_inmem
- SalesLT.SalesOrderHeader\_inmem
- SalesLT.SalesOrderDetail\_inmem
- Demo.DemoSalesOrderHeaderSeed
- Demo.DemoSalesOrderDetailSeed

You can inspect memory-optimized tables through the **Object Explorer** in SSMS. Right-click **Tables** > **Filter** > **Filter Settings** > **Is Memory Optimized**. The value equals 1.

Or you can query the catalog views, such as:

```
SELECT is_memory_optimized, name, type_desc, durability_desc
FROM sys.tables
WHERE is_memory_optimized = 1;
```

**Natively compiled stored procedure:** You can inspect SalesLT.usp\_InsertSalesOrder\_inmem through a catalog view query:

```
SELECT uses_native_compilation, OBJECT_NAME(object_id), definition
FROM sys.sql_modules
WHERE uses_native_compilation = 1;
```

#### Run the sample OLTP workload

The only difference between the following two *stored procedures* is that the first procedure uses memory-optimized versions of the tables, while the second procedure uses the regular on-disk tables:

- SalesLT\*\*.**usp\_InsertSalesOrder\_inmem**\*\*
- SalesLT\*\*.**usp\_InsertSalesOrder\_ondisk**\*\*

In this section, you see how to use the handy **ostress.exe** utility to execute the two stored procedures at stressful levels. You can compare how long it takes for the two stress runs to finish.

When you run ostress.exe, we recommend that you pass parameter values designed for both of the following:

- Run a large number of concurrent connections, by using -n100.
- Have each connection loop hundreds of times, by using -r500.

However, you might want to start with much smaller values like -n10 and -r50 to ensure that everything is working.

#### Script for ostress.exe

This section displays the T-SQL script that is embedded in our ostress.exe command line. The script uses items that were created by the T-SQL script that you installed earlier.

The following script inserts a sample sales order with five line items into the following memory-optimized *tables*:

- SalesLT.SalesOrderHeader\_inmem
- SalesLT.SalesOrderDetail\_inmem

```

DECLARE
 @i int = 0,
 @od SalesLT.SalesOrderDetailType_inmem,
 @SalesOrderID int,
 @DueDate datetime2 = sysdatetime(),
 @CustomerID int = rand() * 8000,
 @BillToAddressID int = rand() * 10000,
 @ShipToAddressID int = rand() * 10000;

INSERT INTO @od
SELECT OrderQty, ProductID
FROM Demo.DemoSalesOrderDetailSeed
WHERE OrderID= cast((rand()*60) as int);

WHILE (@i < 20)
begin;
 EXECUTE SalesLT.usp_InsertSalesOrder_inmem @SalesOrderID OUTPUT,
 @DueDate, @CustomerID, @BillToAddressID, @ShipToAddressID, @od;
 SET @i = @i + 1;
end

```

To make the `_ondisk` version of the preceding T-SQL script for `ostress.exe`, you would replace both occurrences of the `_inmem` substring with `_ondisk`. These replacements affect the names of tables and stored procedures.

#### Install RML utilities and `ostress`

Ideally, you would plan to run `ostress.exe` on an Azure virtual machine (VM). You would create an [Azure VM](#) in the same Azure geographic region where your `AdventureWorksLT` database resides. But you can run `ostress.exe` on your laptop instead.

On the VM, or on whatever host you choose, install the Replay Markup Language (RML) utilities. The utilities include `ostress.exe`.

For more information, see:

- The `ostress.exe` discussion in [Sample Database for In-Memory OLTP](#).
- [Sample Database for In-Memory OLTP](#).
- The [blog for installing ostress.exe](#).

#### Run the `_inmem` stress workload first

You can use an *RML Cmd Prompt* window to run our `ostress.exe` command line. The command-line parameters direct `ostress` to:

- Run 100 connections concurrently (`-n100`).
- Have each connection run the T-SQL script 50 times (`-r50`).

```

ostress.exe -n100 -r50 -S<servername>.database.windows.net -U<login> -P<password> -d<database> -q -Q"DECLARE
@i int = 0, @od SalesLT.SalesOrderDetailType_inmem, @SalesOrderID int, @DueDate datetime2 = sysdatetime(),
@CustomerID int = rand() * 8000, @BillToAddressID int = rand() * 10000, @ShipToAddressID int = rand()* 10000;
INSERT INTO @od SELECT OrderQty, ProductID FROM Demo.DemoSalesOrderDetailSeed WHERE OrderID= cast((rand()*60)
as int); WHILE (@i < 20) begin; EXECUTE SalesLT.usp_InsertSalesOrder_inmem @SalesOrderID OUTPUT, @DueDate,
@CustomerID, @BillToAddressID, @ShipToAddressID, @od; set @i += 1; end"

```

To run the preceding `ostress.exe` command line:

1. Reset the database data content by running the following command in SSMS, to delete all the data that was inserted by any previous runs:

```
EXECUTE Demo.usp_DemoReset;
```

2. Copy the text of the preceding `ostress.exe` command line to your clipboard.
3. Replace the `<placeholders>` for the parameters `-S -U -P -d` with the correct real values.
4. Run your edited command line in an RML Cmd window.

#### Result is a duration

When `ostress.exe` finishes, it writes the run duration as its final line of output in the RML Cmd window. For example, a shorter test run lasted about 1.5 minutes:

```
11/12/15 00:35:00.873 [0x000030A8] OSTRESS exiting normally, elapsed time: 00:01:31.867
```

#### Reset, edit for `_ondisk`, then rerun

After you have the result from the `_inmem` run, perform the following steps for the `_ondisk` run:

1. Reset the database by running the following command in SSMS to delete all the data that was inserted by the previous run:

```
EXECUTE Demo.usp_DemoReset;
```

2. Edit the `ostress.exe` command line to replace all `_inmem` with `_ondisk`.
3. Rerun `ostress.exe` for the second time, and capture the duration result.
4. Again, reset the database (for responsibly deleting what can be a large amount of test data).

#### Expected comparison results

Our In-Memory tests have shown that performance improved by **nine times** for this simplistic workload, with `ostress` running on an Azure VM in the same Azure region as the database.

## 2. Install the In-Memory Analytics sample

In this section, you compare the IO and statistics results when you're using a columnstore index versus a traditional b-tree index.

For real-time analytics on an OLTP workload, it's often best to use a nonclustered columnstore index. For details, see [Columnstore Indexes Described](#).

#### Prepare the columnstore analytics test

1. Use the Azure portal to create a fresh AdventureWorksLT database from the sample.
  - Use that exact name.
  - Choose any Premium service tier.
2. Copy the [sql\\_in-memory\\_analytics\\_sample](#) to your clipboard.
  - The T-SQL script creates the necessary In-Memory objects in the AdventureWorksLT sample database that you created in step 1.
  - The script creates the Dimension table and two fact tables. The fact tables are populated with 3.5 million rows each.
  - The script might take 15 minutes to complete.
3. Paste the T-SQL script into SSMS, and then execute the script. The **COLUMNSTORE** keyword in the **CREATE INDEX** statement is crucial, as in:

```
CREATE NONCLUSTERED COLUMNSTORE INDEX ...;
```

4. Set AdventureWorksLT to compatibility level 130:

```
ALTER DATABASE AdventureworksLT SET compatibility_level = 130;
```

Level 130 is not directly related to In-Memory features. But level 130 generally provides faster query performance than 120.

#### Key tables and columnstore indexes

- dbo.FactResellerSalesXL\_CCI is a table that has a clustered columnstore index, which has advanced compression at the *data* level.
- dbo.FactResellerSalesXL\_PageCompressed is a table that has an equivalent regular clustered index, which is compressed only at the *page* level.

#### Key queries to compare the columnstore index

There are [several T-SQL query types that you can run](#) to see performance improvements. In step 2 in the T-SQL script, pay attention to this pair of queries. They differ only on one line:

- `FROM FactResellerSalesXL_PageCompressed a`
- `FROM FactResellerSalesXL_CCI a`

A clustered columnstore index is in the FactResellerSalesXL\_CCI table.

The following T-SQL script excerpt prints statistics for IO and TIME for the query of each table.

```

*****Step 2 -- Overview
-- Page Compressed BTREE table v/s Columnstore table performance differences
-- Enable actual Query Plan in order to see Plan differences when Executing
*/
-- Ensure Database is in 130 compatibility mode
ALTER DATABASE AdventureworksLT SET compatibility_level = 130
GO

-- Execute a typical query that joins the Fact Table with dimension tables
-- Note this query will run on the Page Compressed table, Note down the time
SET STATISTICS IO ON
SET STATISTICS TIME ON
GO

SELECT c.Year
,e.ProductCategoryKey
,FirstName + ' ' + LastName AS FullName
,count(SalesOrderNumber) AS NumSales
,sum(SalesAmount) AS TotalSalesAmt
,Avg(SalesAmount) AS AvgSalesAmt
,count(DISTINCT SalesOrderNumber) AS NumOrders
,count(DISTINCT a.CustomerKey) AS CountCustomers
FROM FactResellerSalesXL_PageCompressed a
INNER JOIN DimProduct b ON b.ProductKey = a.ProductKey
INNER JOIN DimCustomer d ON d.CustomerKey = a.CustomerKey
Inner JOIN DimProductSubCategory e on e.ProductSubcategoryKey = b.ProductSubcategoryKey
INNER JOIN DimDate c ON c.DateKey = a.OrderDateKey
GROUP BY e.ProductCategoryKey,c.Year,d.CustomerKey,d.FirstName,d.LastName
GO
SET STATISTICS IO OFF
SET STATISTICS TIME OFF
GO

-- This is the same Prior query on a table with a clustered columnstore index CCI
-- The comparison numbers are even more dramatic the larger the table is (this is an 11 million row table
only)
SET STATISTICS IO ON
SET STATISTICS TIME ON
GO
SELECT c.Year
,e.ProductCategoryKey
,FirstName + ' ' + LastName AS FullName
,count(SalesOrderNumber) AS NumSales
,sum(SalesAmount) AS TotalSalesAmt
,Avg(SalesAmount) AS AvgSalesAmt
,count(DISTINCT SalesOrderNumber) AS NumOrders
,count(DISTINCT a.CustomerKey) AS CountCustomers
FROM FactResellerSalesXL_CCI a
INNER JOIN DimProduct b ON b.ProductKey = a.ProductKey
INNER JOIN DimCustomer d ON d.CustomerKey = a.CustomerKey
Inner JOIN DimProductSubCategory e on e.ProductSubcategoryKey = b.ProductSubcategoryKey
INNER JOIN DimDate c ON c.DateKey = a.OrderDateKey
GROUP BY e.ProductCategoryKey,c.Year,d.CustomerKey,d.FirstName,d.LastName
GO

SET STATISTICS IO OFF
SET STATISTICS TIME OFF
GO

```

In a database with the P2 pricing tier, you can expect about nine times the performance gain for this query by using the clustered columnstore index compared with the traditional index. With P15, you can expect about 57 times the performance gain by using the columnstore index.

## Next steps

- [Quickstart 1: In-Memory OLTP Technologies for faster T-SQL Performance](#)
- [Use In-Memory OLTP in an existing Azure SQL application](#)
- [Monitor In-Memory OLTP storage for In-Memory OLTP](#)

## Additional resources

### Deeper information

- [Learn how Quorum doubles key database's workload while lowering DTU by 70% with In-Memory OLTP in SQL Database](#)
- [In-Memory OLTP in Azure SQL Database Blog Post](#)
- [Learn about In-Memory OLTP](#)
- [Learn about columnstore indexes](#)
- [Learn about real-time operational analytics](#)
- See [Common Workload Patterns and Migration Considerations](#) (which describes workload patterns where In-Memory OLTP commonly provides significant performance gains)

### Application design

- [In-Memory OLTP \(In-Memory Optimization\)](#)
- [Use In-Memory OLTP in an existing Azure SQL application](#)

### Tools

- [Azure portal](#)
- [SQL Server Management Studio \(SSMS\)](#)
- [SQL Server Data Tools \(SSDT\)](#)

# Enable automatic tuning to monitor queries and improve workload performance

12/3/2019 • 4 minutes to read • [Edit Online](#)

Azure SQL Database is an automatically managed data service that constantly monitors your queries and identifies the action that you can perform to improve performance of your workload. You can review recommendations and manually apply them, or let Azure SQL Database automatically apply corrective actions - this is known as **automatic tuning mode**.

Automatic tuning can be enabled at the server or the database level through the [Azure portal](#), [REST API](#) calls and [T-SQL](#) commands.

## NOTE

For Managed Instance, the supported option FORCE\_LAST\_GOOD\_PLAN can be configured through [T-SQL](#) only. Portal based configuration and automatic index tuning options described in this article do not apply to Managed Instance.

## NOTE

Configuring Automatic tuning options through ARM (Azure Resource Manager) template is not supported at this time.

## Enable automatic tuning on server

On the server level you can choose to inherit automatic tuning configuration from "Azure Defaults" or not to inherit the configuration. Azure defaults are FORCE\_LAST\_GOOD\_PLAN is enabled, CREATE\_INDEX is enabled, and DROP\_INDEX is disabled.

### Azure portal

To enable automatic tuning on Azure SQL Database logical **server**, navigate to the server in Azure portal and then select **Automatic tuning** in the menu.

OPTION	DESIRED STATE	CURRENT STATE
FORCE PLAN	ON OFF INHERIT	OFF Forced by user
CREATE INDEX	ON OFF INHERIT	OFF Forced by user
DROP INDEX	ON OFF INHERIT	OFF Forced by user

The selected configuration will be applied to all the databases that inherit automatic tuning configuration from this server. [Click to see the list of databases.](#)

[Apply](#)

#### NOTE

Please note that **DROP\_INDEX** option at this time is not compatible with applications using partition switching and index hints and should not be enabled in these cases. Dropping unused indexes is not supported for Premium and Business Critical service tiers.

Select the automatic tuning options you want to enable and select **Apply**.

Automatic tuning options on a server are applied to all databases on this server. By default, all databases inherit configuration from their parent server, but this can be overridden and specified for each database individually.

#### REST API

Find out more about using REST API to enable Automatic tuning on a server, see [SQL Server Automatic tuning UPDATE and GET HTTP methods](#).

## Enable automatic tuning on an individual database

The Azure SQL Database enables you to individually specify the automatic tuning configuration for each database. On the database level you can choose to inherit automatic tuning configuration from the parent server, "Azure Defaults" or not to inherit the configuration. Azure Defaults are set to FORCE\_LAST\_GOOD\_PLAN is enabled, CREATE\_INDEX is enabled, and DROP\_INDEX is disabled.

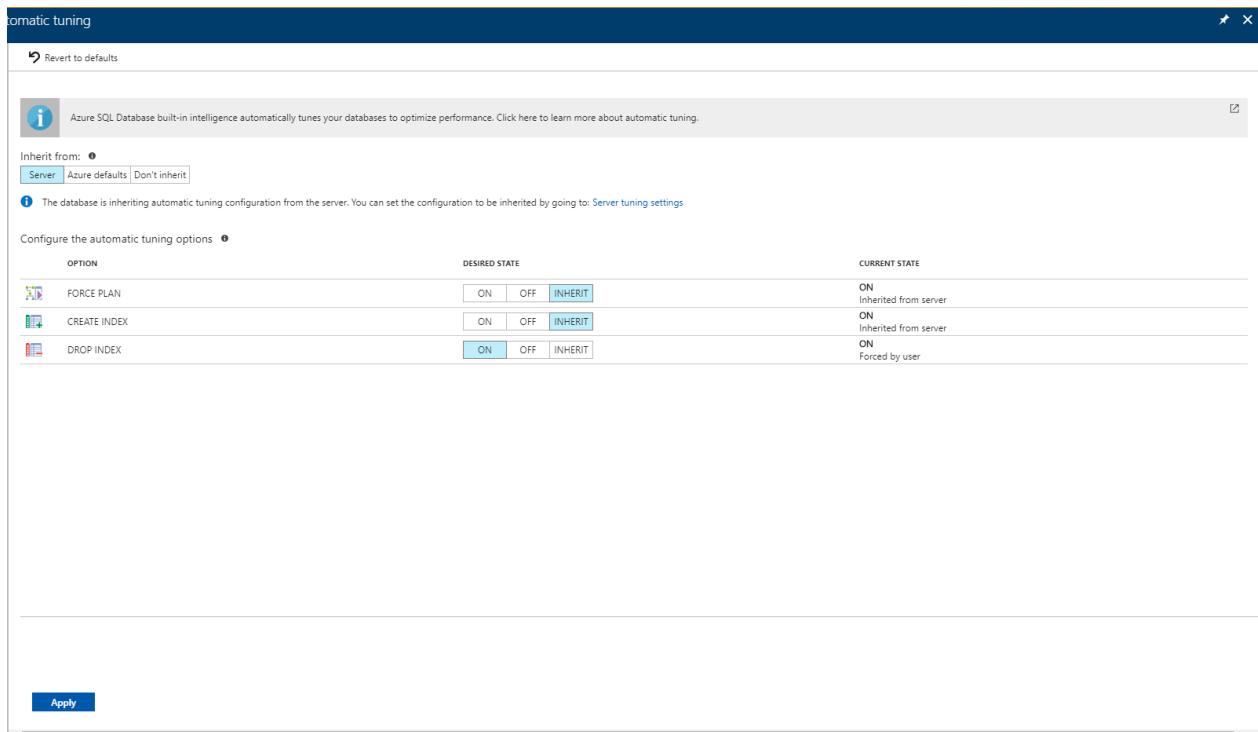
#### TIP

The general recommendation is to manage the automatic tuning configuration at **server level** so the same configuration settings can be applied on every database automatically. Configure automatic tuning on an individual database only if you need that database to have different settings than others inheriting settings from the same server.

#### Azure portal

To enable automatic tuning on a **single database**, navigate to the database in Azure portal and select **Automatic tuning**.

Individual automatic tuning settings can be separately configured for each database. You can manually configure an individual automatic tuning option, or specify that an option inherits its settings from the server.



Please note that DROP\_INDEX option at this time is not compatible with applications using partition switching and index hints and should not be enabled in these cases.

Once you have selected your desired configuration, click **Apply**.

## Rest API

Find out more about using REST API to enable Automatic tuning on a single database, see [SQL Database Automatic tuning UPDATE and GET HTTP methods](#).

## T-SQL

To enable automatic tuning on a single database via T-SQL, connect to the database and execute the following query:

```
ALTER DATABASE current SET AUTOMATIC_TUNING = AUTO | INHERIT | CUSTOM
```

Setting automatic tuning to AUTO will apply Azure Defaults. Setting it to INHERIT, automatic tuning configuration will be inherited from the parent server. Choosing CUSTOM, you will need to manually configure automatic tuning.

To configure individual automatic tuning options via T-SQL, connect to the database and execute the query such as this one:

```
ALTER DATABASE current SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON, CREATE_INDEX = DEFAULT, DROP_INDEX = OFF)
```

Setting the individual tuning option to ON, will override any setting that database inherited and enable the tuning option. Setting it to OFF, will also override any setting that database inherited and disable the tuning option. Automatic tuning option, for which DEFAULT is specified, will inherit the automatic tuning configuration from the server level settings.

### IMPORTANT

In case of [active geo-replication](#), Automatic tuning needs to be configured on the primary database only. Automatically applied tuning actions, such as for example index create or delete will be automatically replicated to the read-only secondary. Attempting to enable Automatic tuning via T-SQL on the read-only secondary will result in a failure as having a different tuning configuration on the read-only secondary is unsupported.

Find our more abut T-SQL options to configure Automatic tuning, see [ALTER DATABASE SET Options \(Transact-SQL\) for SQL Database server](#).

## Disabled by the system

Automatic tuning is monitoring all the actions it takes on the database and in some cases it can determine that automatic tuning can't properly work on the database. In this situation, tuning option will be disabled by the system. In most cases this happens because Query Store is not enabled or it's in read-only state on a specific database.

## Permissions

As automatic tuning is Azure feature, to use it you will need to use Azure's built-in RBAC roles. Using SQL Authentication only will not be sufficient to use the feature from Azure portal.

To use automatic tuning, the minimum required permission to grant to the user is Azure's built-in [SQL DB](#)

[contributor](#) role. You can also consider using higher privilege roles such are SQL Server Contributor, Contributor and Owner.

## Configure automatic tuning e-mail notifications

See [automatic tuning e-mail notifications](#) guide.

## Next steps

- Read the [Automatic tuning article](#) to learn more about automatic tuning and how it can help you improve your performance.
- See [Performance recommendations](#) for an overview of Azure SQL Database performance recommendations.
- See [Query Performance Insights](#) to learn about viewing the performance impact of your top queries.

# Email notifications for automatic tuning

11/7/2019 • 10 minutes to read • [Edit Online](#)

SQL Database tuning recommendations are generated by Azure SQL Database [Automatic tuning](#). This solution continuously monitors and analyzes workloads of SQL Databases providing customized tuning recommendations for each individual database related to index creation, index deletion, and optimization of query execution plans.

SQL Database Automatic tuning recommendations can be viewed in the [Azure portal](#), retrieved with [REST API](#) calls, or by using [T-SQL](#) and [PowerShell](#) commands. This article is based on using a PowerShell script to retrieve automatic tuning recommendations.

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

## Automate email notifications for Automatic tuning recommendations

The following solution automates the sending of email notifications containing Automatic tuning recommendations. The solution described consists of automating execution of a PowerShell script for retrieving tuning recommendations using [Azure Automation](#), and automation of scheduling email delivery job using [Microsoft Flow](#).

## Create Azure Automation account

To use Azure Automation, the first step is to create an automation account and to configure it with Azure resources to use for execution of the PowerShell script. To learn more about Azure Automation and its capabilities, see [Getting started with Azure automation](#).

Follow these steps to create Azure Automation Account through the method of selecting and configuring Automation app from the Marketplace:

- Log into the Azure portal
- Click on “**+ Create a resource**” in the upper left corner
- Search for “**Automation**” (press enter)
- Click on the Automation app in the search results

NAME	PUBLISHER	CATEGORY
Automation	Microsoft	Developer tools
Automation & Control	Microsoft	Monitoring + Manage...
Chef Automate	Chef Software, Inc	Compute

- Once inside the "Create an Automation Account" pane, click on "**Create**"
- Populate the required information: enter a name for this automation account, select your Azure subscription ID and Azure resources to be used for the PowerShell script execution
- For the "**Create Azure Run As account**" option, select **Yes** to configure the type of account under which PowerShell script runs with the help of Azure Automation. To learn more about account types, see [Run As account](#)
- Conclude creation of the automation account by clicking on **Create**

#### TIP

Record your Azure Automation account name, subscription ID, and resources (such as copy-paste to a notepad) exactly as entered while creating the Automation app. You need this information later.

If you have several Azure subscriptions for which you would like to build the same automation, you need to repeat this process for your other subscriptions.

## Update Azure Automation modules

The PowerShell script to retrieve Automatic tuning recommendation uses [Get-AzResource](#) and [Get-AzSqlDatabaseRecommendedAction](#) commands for which Azure Module version 4 and above is required.

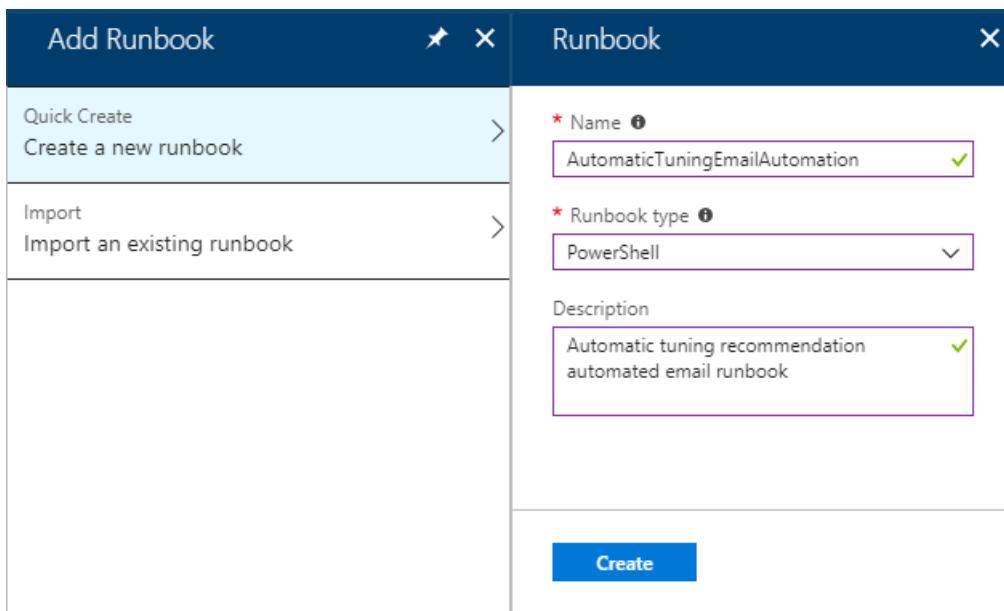
- In case your Azure Modules need updating, see [Az module support in Azure Automation](#).

## Create Azure Automation Runbook

The next step is to create a Runbook in Azure Automation inside which the PowerShell script for retrieval of tuning recommendations resides.

Follow these steps to create a new Azure Automation runbook:

- Access the Azure Automation account you created in the previous step
- Once in the automation account pane, click on the "**Runbooks**" menu item on the left-hand side to create a new Azure Automation runbook with the PowerShell script. To learn more about creating automation runbooks, see [Creating a new runbook](#).
- To add a new runbook, click on the "**+Add a runbook**" menu option, and then click on the "**Quick create – Create a new runbook**".
- In the Runbook pane, type in the name of your runbook (for the purpose of this example, "**AutomaticTuningEmailAutomation**" is used), select the type of runbook as **PowerShell** and write a description of this runbook to describe its purpose.
- Click on the **Create** button to finish creating a new runbook



Follow these steps to load a PowerShell script inside the runbook created:

- Inside the “**Edit PowerShell Runbook**” pane, select “**RUNBOOKS**” on the menu tree and expand the view until you see the name of your runbook (in this example “**AutomaticTuningEmailAutomation**”). Select this runbook.
- On the first line of the “Edit PowerShell Runbook” (starting with the number 1), copy-paste the following PowerShell script code. This PowerShell script is provided as-is to get you started. Modify the script to suite your needs.

In the header of the provided PowerShell script, you need to replace `<SUBSCRIPTION_ID_WITH_DATABASES>` with your Azure subscription ID. To learn how to retrieve your Azure subscription ID, see [Getting your Azure Subscription GUID](#).

In case of several subscriptions, you can add them as comma-delimited to the “\$subscriptions” property in the header of the script.

```
PowerShell script to retrieve Azure SQL Database Automatic tuning recommendations.
#
Provided "as-is" with no implied warranties or support.
The script is released to the public domain.
#
Replace <SUBSCRIPTION_ID_WITH_DATABASES> in the header with your Azure subscription ID.
#
Microsoft Azure SQL Database team, 2018-01-22.

Set subscriptions : IMPORTANT - REPLACE <SUBSCRIPTION_ID_WITH_DATABASES> WITH YOUR SUBSCRIPTION ID
$subscriptions = ("<SUBSCRIPTION_ID_WITH_DATABASES>", "<SECOND_SUBSCRIPTION_ID_WITH_DATABASES>", "<THIRD_SUBSCRIPTION_ID_WITH_DATABASES>")

Get credentials
$Conn = Get-AutomationConnection -Name AzureRunAsConnection
Connect-AzAccount -ServicePrincipal -Tenant $Conn.TenantID -ApplicationId $Conn.ApplicationID -
CertificateThumbprint $Conn.CertificateThumbprint

Define the resource types
$resourceTypes = ("Microsoft.Sql/servers/databases")
$advisors = ("CreateIndex", "DropIndex");
$results = @()

Loop through all subscriptions
foreach($subscriptionId in $subscriptions) {
 Select-AzSubscription -SubscriptionId $subscriptionId
 $rgs = Get-AzResourceGroup
```

```

Loop through all resource groups
foreach($rg in $rgs) {
 $rgname = $rg.ResourceGroupName;

 # Loop through all resource types
 foreach($resourceType in $resourceTypes) {
 $resources = Get-AzResource -ResourceGroupName $rgname -ResourceType $resourceType

 # Loop through all databases
 # Extract resource groups, servers and databases
 foreach ($resource in $resources) {
 $resourceId = $resource.ResourceId
 if ($resourceId -match ".*RESOURCEGROUPS/(?<content>.*)/PROVIDERS.*") {
 $ResourceGroupName = $matches['content']
 } else {
 continue
 }
 if ($resourceId -match ".*SERVERS/(?<content>.*)/DATABASES.*") {
 $ServerName = $matches['content']
 } else {
 continue
 }
 if ($resourceId -match ".*/DATABASES/(?<content>.*)") {
 $DatabaseName = $matches['content']
 } else {
 continue
 }

 # Skip if master
 if ($DatabaseName -eq "master") {
 continue
 }

 # Loop through all Automatic tuning recommendation types
 foreach ($advisor in $advisors) {
 $recs = Get-AzSqlDatabaseRecommendedAction -ResourceGroupName $ResourceGroupName -
ServerName $ServerName -DatabaseName $DatabaseName -AdvisorName $advisor
 foreach ($r in $recs) {
 if ($r.State.CurrentValue -eq "Active") {
 $object = New-Object -TypeName PSObject
 $object | Add-Member -Name 'SubscriptionId' -MemberType NoteProperty -Value
$subscriptionId
 $object | Add-Member -Name 'ResourceGroupName' -MemberType NoteProperty -Value
$rg.ResourceGroupName
 $object | Add-Member -Name 'ServerName' -MemberType NoteProperty -Value
$r.ServerName
 $object | Add-Member -Name 'DatabaseName' -MemberType NoteProperty -Value
$r.DatabaseName
 $object | Add-Member -Name 'Script' -MemberType NoteProperty -Value
$r.ImplementationDetails.Script
 $results += $object
 }
 }
 }
 }
 }

 # Format and output results for the email
 $table = $results | Format-List
 Write-Output $table
}

```

Click the “**Save**” button in the upper right corner to save the script. When you are satisfied with the script, click the “**Publish**” button to publish this runbook.

At the main runbook pane, you can choose to click on the “**Start**” button to **test** the script. Click on the “**Output**”

to view results of the script executed. This output is going to be the content of your email. The sample output from the script can be seen in the following screenshot.

The screenshot shows the Azure Automation Job Output pane. On the left, there's a summary of the job: Job ID, Created, Job status, Completed, Run As, User, Ran on, and Source snapshot. Below that is an Overview section with tabs for Input (0), Output (highlighted in blue), and All Logs. Under Errors, there are 0 errors (red X). Under Warnings, there are 0 warnings (yellow triangle). The main pane displays the PowerShell script output:

```
Environments

{[AzureUSGovernment, AzureUSGovernment], [AzureChinaCloud, AzureChinaCloud], [AzureGermanCloud, AzureGermanCloud], [A...}

Name :
Account :
Environment : AzureCloud
Subscription :
Tenant :
TokenCache : Microsoft.Azure.Commands.Common.Authentication.AuthenticationStoreTokenCache
VersionProfile :
ExtendedProperties : {}

SubscriptionId :
ResourceGroupName :
ServerName :
DatabaseName :
Script : DROP INDEX [] ON [dbo].[test_hinted_drop_name]

SubscriptionId :
ResourceGroupName :
ServerName :
DatabaseName :
Script : CREATE NONCLUSTERED INDEX [] ON [CRM].[DataPoints] ([Name],[Money],[Power]) INCLUDE ([Hour], [System], [LastChanged]) WITH (ONLINE = ON)

SubscriptionId :
ResourceGroupName :
ServerName :
DatabaseName :
Script : CREATE NONCLUSTERED INDEX [] ON [dbo].[Employees] ([City], [State]) INCLUDE ([Postal]) WITH (ONLINE = ON)
```

Ensure to adjust the content by customizing the PowerShell script to your needs.

With the above steps, the PowerShell script to retrieve Automatic tuning recommendations is loaded in Azure Automation. The next step is to automate and schedule the email delivery job.

## Automate the email jobs with Microsoft Flow

To complete the solution, as the final step, create an automation flow in Microsoft Flow consisting of three actions (jobs):

1. **"Azure Automation - Create job"** – used to execute the PowerShell script to retrieve Automatic tuning recommendations inside the Azure Automation runbook
2. **"Azure Automation - Get job output"** – used to retrieve output from the executed PowerShell script
3. **"Office 365 Outlook - Send an email"** – used to send out email. E-mails are sent out using the Office 365 account of the individual creating the flow.

To learn more about Microsoft Flow capabilities, see [Getting started with Microsoft Flow](#).

Prerequisite for this step is to sign up for [Microsoft Flow](#) account and to log in. Once inside the solution, follow these steps to set up a **new flow**:

- Access "**My flows**" menu item
- Inside My flows, select the "**+Create from blank**" link at the top of the page
- Click on the link "**Search for hundreds of connectors and triggers**" at the bottom of the page
- In the search field type "**recurrence**", and select "**Schedule - Recurrence**" from the search results to schedule the email delivery job to run.
- In the Recurrence pane in the Frequency field, select the scheduling frequency for this flow to execute, such as send automated email each Minute, Hour, Day, Week, etc.

The next step is to add three jobs (create, get output and send email) to the newly created recurring flow. To accomplish adding the required jobs to the flow, follow these steps:

1. Create action to execute PowerShell script to retrieve tuning recommendations

- Select “**+New step**”, followed by “**Add an action**” inside the Recurrence flow pane
  - In the search field type “**automation**” and select “**Azure Automation – Create job**” from the search results
  - In the Create job pane, configure the job properties. For this configuration, you will need details of your Azure subscription ID, Resource Group and Automation Account **previously recorded** at the **Automation Account pane**. To learn more about options available in this section, see [Azure Automation - Create Job](#).
  - Complete creating this action by clicking on “**Save flow**”
2. Create action to retrieve output from the executed PowerShell script
- Select “**+New step**”, followed by “**Add an action**” inside the Recurrence flow pane
  - In the search filed type “**automation**” and select “**Azure Automation – Get job output**” from the search results. To learn more about options available in this section, see [Azure Automation – Get job output](#).
  - Populate fields required (similar to creating the previous job) - populate your Azure subscription ID, Resource Group, and Automation Account (as entered in the Automation Account pane)
  - Click inside the field “**Job ID**” for the “**Dynamic content**” menu to show up. From within this menu, select the option “**Job ID**”.
  - Complete creating this action by clicking on “**Save flow**”

3. Create action to send out email using Office 365 integration

- Select “**+New step**”, followed by “**Add an action**” inside the Recurrence flow pane
- In the search filed type “**send an email**” and select “**Office 365 Outlook – Send an email**” from the search results
- In the “**To**” field type in the email address to which you need to send the notification email
- In the “**Subject**” field type in the subject of your email, for example “Automatic tuning recommendations email notification”
- Click inside the field “**Body**” for the “**Dynamic content**” menu to show up. From within this menu, under “**Get job output**”, select “**Content**”
- Complete creating this action by clicking on “**Save flow**”

**TIP**

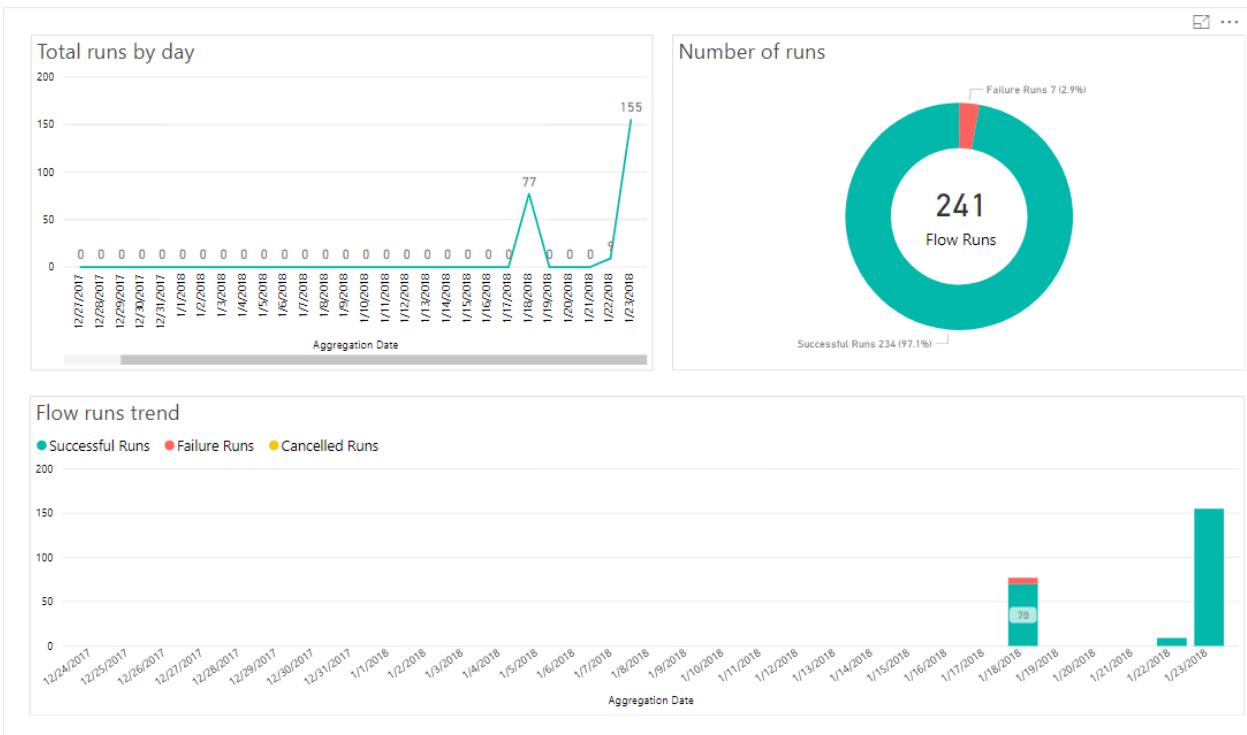
To send automated emails to different recipients, create separate flows. In these additional flows, change the recipient email address in the “To” field, and the email subject line in the “Subject” field. Creating new runbooks in Azure Automation with customized PowerShell scripts (such as with change of Azure subscription ID) enables further customization of automated scenarios, such is for example emailing separate recipients on Automated tuning recommendations for separate subscriptions.

The above concludes steps required to configure the email delivery job workflow. The entire flow consisting of three actions built is shown in the following image.



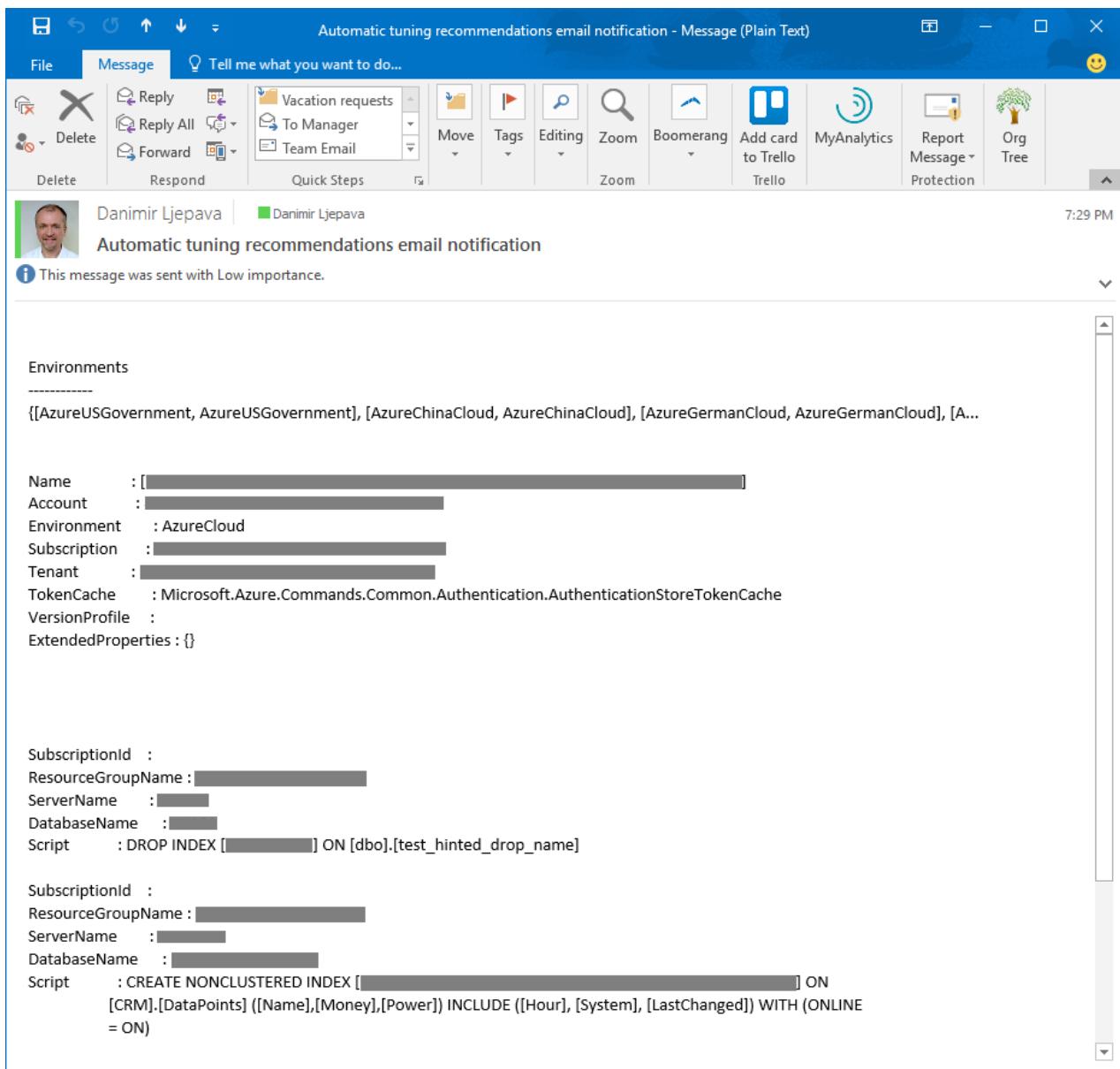
To test the flow, click on “Run Now” in the upper right corner inside the flow pane.

Statistics of running the automated jobs, showing success of email notifications sent out, can be seen from the Flow analytics pane.



The Flow analytics is helpful for monitoring the success of job executions, and if required for troubleshooting. In the case of troubleshooting, you also might want to examine the PowerShell script execution log accessible through Azure Automation app.

The final output of the automated email looks similar to the following email received after building and running this solution:



By adjusting the PowerShell script, you can adjust the output and formatting of the automated email to your needs.

You might further customize the solution to build email notifications based on a specific tuning event, and to multiple recipients, for multiple subscriptions or databases, depending on your custom scenarios.

## Next steps

- Learn more on how automatic tuning can help you improve database performance, see [Automatic tuning in Azure SQL Database](#).
- To enable automatic tuning in Azure SQL Database to manage your workload, see [Enable automatic tuning](#).
- To manually review and apply Automatic tuning recommendations, see [Find and apply performance recommendations](#).

# Find and apply performance recommendations

11/7/2019 • 5 minutes to read • [Edit Online](#)

You can use the Azure portal to find performance recommendations that can optimize performance of your Azure SQL Database or to correct some issue identified in your workload. **Performance recommendation** page in Azure portal enables you to find the top recommendations based on their potential impact.

## Viewing recommendations

To view and apply performance recommendations, you need the correct [role-based access control](#) permissions in Azure. **Reader, SQL DB Contributor** permissions are required to view recommendations, and **Owner, SQL DB Contributor** permissions are required to execute any actions; create or drop indexes and cancel index creation.

Use the following steps to find performance recommendations on Azure portal:

1. Sign in to the [Azure portal](#).
2. Go to **All services > SQL databases**, and select your database.
3. Navigate to **Performance recommendation** to view available recommendations for the selected database.

Performance recommendations are shown in the table similar to the one shown on the following figure:

Recommendations

ACTION	RECOMMENDATION DESCRIPTION	IMPACT
CREATE INDEX	Table: [test_table_0.430709] Indexed columns: [index_1],[index_2],[index_3]	HIGH IMPACT
CREATE INDEX	Table: [test_table_0.914675] Indexed columns: [index_1],[index_2],[index_3]	HIGH IMPACT
DROP INDEX (PREVIEW)	Index name: IR_[test_schema]_[test_table_0.112348]_CD2E5085881888FC9A4' Reason: Duplicate index	HIGH IMPACT
DROP INDEX (PREVIEW)	Index name: IR_[test_schema]_[test_table_0.950691]_9A67D9E88A31B315D14 Reason: Duplicate index	HIGH IMPACT
FIX SCHEMA ISSUES (PREVIEW)	Error code: 208 Error message: Invalid object name 'dbo.Companies'.	HIGH IMPACT

Recommendations are sorted by their potential impact on performance into the following categories:

IMPACT	DESCRIPTION
High	High impact recommendations should provide the most significant performance impact.
Medium	Medium impact recommendations should improve performance, but not substantially.
Low	Low impact recommendations should provide better performance than without, but improvements might not be significant.

#### NOTE

Azure SQL Database needs to monitor activities at least for a day in order to identify some recommendations. The Azure SQL Database can more easily optimize for consistent query patterns than it can for random spotty bursts of activity. If recommendations are not currently available, the **Performance recommendation** page provides a message explaining why.

You can also view the status of the historical operations. Select a recommendation or status to see more information.

Here is an example of "Create index" recommendation in the Azure portal.

The screenshot shows a modal window titled "Create index" for the "[dbo].[Employees]" table. At the top, there are "Apply" and "Discard" buttons, and a "View script" link. Below this, a table displays the recommended action:

RECOMMENDED ACTION	STATUS	LAST UPDATE	INITIATED BY
Create index <a href="#">Learn more</a>	Active ⓘ	5/19/2017 2:42:12 PM	N/A

Below the table, the "Estimated impact" section shows:

Impact ⓘ	HIGH
Disk space needed ⓘ	128.00 MB

The "Details" section lists the following properties:

Index name ⓘ	nci_wi_Employees_8C18C2AF4267DC777930407826
Index type ⓘ	NONCLUSTERED
Schema ⓘ	[dbo]
Table ⓘ	[Employees]
Index key columns ⓘ	[City], [State]
Included columns ⓘ	[Postal]

## Applying recommendations

Azure SQL Database gives you full control over how recommendations are enabled using any of the following

three options:

- Apply individual recommendations one at a time.
- Enable the Automatic tuning to automatically apply recommendations.
- To implement a recommendation manually, run the recommended T-SQL script against your database.

Select any recommendation to view its details and then click **View script** to review the exact details of how the recommendation is created.

The database remains online while the recommendation is applied -- using performance recommendation or automatic tuning never takes a database offline.

### Apply an individual recommendation

You can review and accept recommendations one at a time.

1. On the **Recommendations** page, select a recommendation.
2. On the **Details** page, click **Apply** button.



Selected recommendation are applied on the database.

### Removing recommendations from the list

If your list of recommendations contains items that you want to remove from the list, you can discard the recommendation:

1. Select a recommendation in the list of **Recommendations** to open the details.
2. Click **Discard** on the **Details** page.

If desired, you can add discarded items back to the **Recommendations** list:

1. On the **Recommendations** page, click **View discarded**.
2. Select a discarded item from the list to view its details.
3. Optionally, click **Undo Discard** to add the index back to the main list of **Recommendations**.

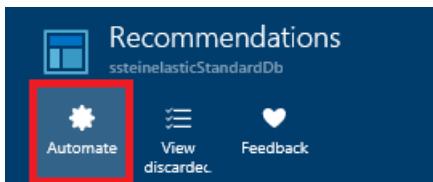
#### NOTE

Please note that if SQL Database [Automatic tuning](#) is enabled, and if you have manually discarded a recommendation from the list, such recommendation will never be applied automatically. Discarding a recommendation is a handy way for users to have Automatic tuning enabled in cases when requiring that a specific recommendation shouldn't be applied. You can revert this behavior by adding discarded recommendations back to the Recommendations list by selecting the Undo Discard option.

### Enable automatic tuning

You can set the Azure SQL Database to implement recommendations automatically. As recommendations become available, they are automatically applied. As with all recommendations managed by the service, if the performance impact is negative, the recommendation is reverted.

1. On the **Recommendations** page, click **Automate**:



## 2. Select actions to automate:

OPTION	DESIRED STATE	CURRENT STATE
FORCE PLAN	ON OFF INHERIT	OFF Forced by user
CREATE INDEX	ON OFF INHERIT	OFF Forced by user
DROP INDEX	ON OFF INHERIT	OFF Forced by user

### NOTE

Please note that **DROP\_INDEX** option is currently not compatible with applications using partition switching and index hints.

Once you have selected your desired configuration, click **Apply**.

### Manually apply recommendations through T-SQL

Select any recommendation and then click **View script**. Run this script against your database to manually apply the recommendation.

*Indexes that are manually executed are not monitored and validated for performance impact by the service so it is suggested that you monitor these indexes after creation to verify they provide performance gains and adjust or delete them if necessary. For details about creating indexes, see [CREATE INDEX \(Transact-SQL\)](#). In addition, manually applied recommendations will remain active and shown in the list of recommendations for 24-48 hrs. before the system automatically withdraws them. If you would like to remove a recommendation sooner, you can manually discard it.*

### Cancelling recommendations

Recommendations that are in a **Pending**, **Validating**, or **Success** status can be canceled. Recommendations with a status of **Executing** cannot be canceled.

1. Select a recommendation in the **Tuning History** area to open the **recommendations details** page.
2. Click **Cancel** to abort the process of applying the recommendation.

## Monitoring operations

Applying a recommendation might not happen instantaneously. The portal provides details regarding the status of recommendation. The following are possible states that an index can be in:

STATUS	DESCRIPTION
Pending	Apply recommendation command has been received and is scheduled for execution.
Executing	The recommendation is being applied.
Validating	Recommendation was successfully applied and the service is measuring the benefits.
Success	Recommendation was successfully applied and benefits have been measured.
Error	An error occurred during the process of applying the recommendation. This can be a transient issue, or possibly a schema change to the table and the script is no longer valid.
Reverting	The recommendation was applied, but has been deemed non-performant and is being automatically reverted.
Reverted	The recommendation was reverted.

Click an in-process recommendation from the list to see more information:

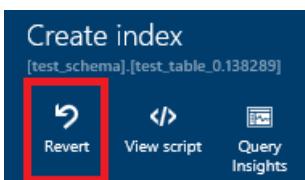
#### Tuning history

ACTION	RECOMMENDATION DESCRIPTION	STATUS	TIME
DROP INDEX (PREVIEW) Initiated by: User	Index name: IR_[test_schema]_[test_table_0.182511]_1665B81581 Reason: Duplicate index	Success	4/25/2016 4:28:05 PM
CREATE INDEX Initiated by: User	Table: [test_table_0.138289] Indexed columns: [index_1],[index_2],[index_3]	Success	4/25/2016 4:27:57 PM
PARAMETERIZE QUERIES (PREVIEW) Initiated by: N/A	Scope: Entire database Reason: Non-parameterized queries are causing performance issues	Success	4/21/2016 4:40:30 PM
DROP INDEX (PREVIEW) Initiated by: User	Index name: IR_[test_schema]_[test_table_0.574879]_C13F85C293 Reason: Duplicate index	Error	4/25/2016 4:28:05 PM

#### Reverting a recommendation

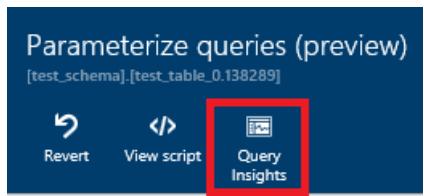
If you used the performance recommendations to apply the recommendation (meaning you did not manually run the T-SQL script), it automatically reverts the change if it finds the performance impact to be negative. If for any reason you simply want to revert a recommendation, you can do the following:

1. Select a successfully applied recommendation in the **Tuning history** area.
2. Click **Revert** on the **recommendation details** page.



## Monitoring performance impact of index recommendations

After recommendations are successfully implemented (currently, index operations and parameterize queries recommendations only), you can click **Query Insights** on the recommendation details page to open [Query Performance Insights](#) and see the performance impact of your top queries.



## Summary

Azure SQL Database provides recommendations for improving SQL database performance. By providing T-SQL scripts, you get assistance in optimizing your database and ultimately improving query performance.

## Next steps

Monitor your recommendations and continue to apply them to refine performance. Database workloads are dynamic and change continuously. Azure SQL Database continues to monitor and provide recommendations that can potentially improve your database's performance.

- See [Automatic tuning](#) to learn more about the automatic tuning in Azure SQL Database.
- See [Performance recommendations](#) for an overview of Azure SQL Database performance recommendations.
- See [Query Performance Insights](#) to learn about viewing the performance impact of your top queries.

## Additional resources

- [Query Store](#)
- [CREATE INDEX](#)
- [Role-based access control](#)

# Create alerts for Azure SQL Database and Data Warehouse using Azure portal

1/8/2020 • 4 minutes to read • [Edit Online](#)

## Overview

This article shows you how to set up Azure SQL Database and Data Warehouse alerts using the Azure portal. Alerts can send you an email or call a web hook when some metric (for example database size or CPU usage) reaches the threshold. This article also provides best practices for setting alert periods.

### IMPORTANT

This feature is not yet available in Managed Instance. As an alternative, you can use SQL Agent to send email alerts for some metrics based on [Dynamic Management Views](#).

You can receive an alert based on monitoring metrics for, or events on, your Azure services.

- **Metric values** - The alert triggers when the value of a specified metric crosses a threshold you assign in either direction. That is, it triggers both when the condition is first met and then afterwards when that condition is no longer being met.
- **Activity log events** - An alert can trigger on *every* event, or, only when a certain number of events occur.

You can configure an alert to do the following when it triggers:

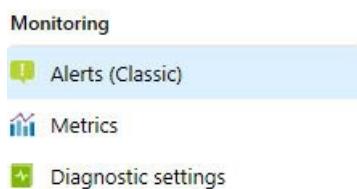
- send email notifications to the service administrator and co-administrators
- send email to additional emails that you specify.
- call a webhook

You can configure and get information about alert rules using

- [Azure portal](#)
- [PowerShell](#)
- [command-line interface \(CLI\)](#)
- [Azure Monitor REST API](#)

## Create an alert rule on a metric with the Azure portal

1. In the [portal](#), locate the resource you are interested in monitoring and select it.
2. Select **Alerts (Classic)** under the MONITORING section. The text and icon may vary slightly for different resources.



- **SQL DW ONLY:** Click the **DWU Usage** graph. Select **View classic alerts**
3. Select the **Add metric alert (classic)** button and fill in the fields.

**Add rule**

\* Name [?](#)  
MyNewDBAlert ✓

Description  
Sample alert for testing ✓

Source  
Alert on  
Metrics

Criteria

Subscription  
\*\*\*\*\*

Resource group  
\*\*\*\*\*

Resource  
\*\*\*\*\* /SampleDB

\* Metric [?](#)  
CPU percentage

Condition  
Greater than

\* Threshold  
60 ✓ %

Period [?](#)  
Over the last 5 minutes

Notify via

Email owners, contributors, and readers

Additional administrator email(s)  
admin@contoso.com ✓

Webhook [?](#)  
http://www.contoso.com/dowork?param ✓

Learn more about configuring webhooks

---

Take action [?](#)  
Run a logic app from this alert >

**OK**

4. **Name** your alert rule, and choose a **Description**, which also shows in notification emails.
5. Select the **Metric** you want to monitor, then choose a **Condition** and **Threshold** value for the metric. Also choose the **Period** of time that the metric rule must be satisfied before the alert triggers. So for example, if you use the period "PT5M" and your alert looks for CPU above 80%, the alert triggers when the **average** CPU has been above 80% for 5 minutes. Once the first trigger occurs, it again triggers when the average CPU is below 80% over 5 minutes. The CPU measurement occurs every 1 minute. Consult the table below

for supported time windows and the aggregation type that each alert uses- not all alerts use the average value.

6. Check **Email owners...** if you want administrators and co-administrators to be emailed when the alert fires.
7. If you want additional emails to receive a notification when the alert fires, add them in the **Additional Administrator email(s)** field. Separate multiple emails with semi-colons -  
*email@contoso.com;email2@contoso.com*
8. Put in a valid URI in the **Webhook** field if you want it called when the alert fires.
9. Select **OK** when done to create the alert.

Within a few minutes, the alert is active and triggers as previously described.

## Managing your alerts

Once you have created an alert, you can select it and:

- View a graph showing the metric threshold and the actual values from the previous day.
- Edit or delete it.
- **Disable** or **Enable** it if you want to temporarily stop or resume receiving notifications for that alert.

## SQL Database alert values

RESOURCE TYPE	METRIC NAME	FRIENDLY NAME	AGGREGATION TYPE	MINIMUM ALERT TIME WINDOW
SQL database	cpu_percent	CPU percentage	Average	5 minutes
SQL database	physical_data_read_percent	Data IO percentage	Average	5 minutes
SQL database	log_write_percent	Log IO percentage	Average	5 minutes
SQL database	dtu_consumption_percent	DTU percentage	Average	5 minutes
SQL database	storage	Total database size	Maximum	30 minutes
SQL database	connection_successful	Successful Connections	Total	10 minutes
SQL database	connection_failed	Failed Connections	Total	10 minutes
SQL database	blocked_by_firewall	Blocked by Firewall	Total	10 minutes
SQL database	deadlock	Deadlocks	Total	10 minutes
SQL database	storage_percent	Database size percentage	Maximum	30 minutes
SQL database	xtp_storage_percent	In-Memory OLTP storage percent(Preview)	Average	5 minutes

Resource Type	Metric Name	Friendly Name	Aggregation Type	Minimum Alert Time Window
SQL database	workers_percent	Workers percentage	Average	5 minutes
SQL database	sessions_percent	Sessions percent	Average	5 minutes
SQL database	dtu_limit	DTU limit	Average	5 minutes
SQL database	dtu_used	DTU used	Average	5 minutes
Elastic pool	cpu_percent	CPU percentage	Average	10 minutes
Elastic pool	physical_data_read_percent	Data IO percentage	Average	10 minutes
Elastic pool	log_write_percent	Log IO percentage	Average	10 minutes
Elastic pool	dtu_consumption_percent	DTU percentage	Average	10 minutes
Elastic pool	storage_percent	Storage percentage	Average	10 minutes
Elastic pool	workers_percent	Workers percentage	Average	10 minutes
Elastic pool	eDTU_limit	eDTU limit	Average	10 minutes
Elastic pool	storage_limit	Storage limit	Average	10 minutes
Elastic pool	eDTU_used	eDTU used	Average	10 minutes
Elastic pool	storage_used	Storage used	Average	10 minutes
SQL data warehouse	cpu_percent	CPU percentage	Average	10 minutes
SQL data warehouse	physical_data_read_percent	Data IO percentage	Average	10 minutes
SQL data warehouse	connection_successful	Successful Connections	Total	10 minutes
SQL data warehouse	connection_failed	Failed Connections	Total	10 minutes
SQL data warehouse	blocked_by_firewall	Blocked by Firewall	Total	10 minutes
SQL data warehouse	service_level_objective	Service tier of the database	Total	10 minutes
SQL data warehouse	dwu_limit	dwu limit	Maximum	10 minutes

RESOURCE TYPE	METRIC NAME	FRIENDLY NAME	AGGREGATION TYPE	MINIMUM ALERT TIME WINDOW
SQL data warehouse	dwu_consumption_percent	DWU percentage	Average	10 minutes
SQL data warehouse	dwu_used	DWU used	Average	10 minutes

## Next steps

- Get an [overview of Azure monitoring](#) including the types of information you can collect and monitor.
- Learn more about [configuring webhooks in alerts](#).
- Get an [overview of diagnostic logs](#) and collect detailed high-frequency metrics on your service.
- Get an [overview of metrics collection](#) to make sure your service is available and responsive.

# Manage file space for single and pooled databases in Azure SQL Database

11/22/2019 • 8 minutes to read • [Edit Online](#)

This article describes different types of storage space for single and pooled databases in Azure SQL Database, and steps that can be taken when the file space allocated for databases and elastic pools needs to be explicitly managed.

## NOTE

This article does not apply to the managed instance deployment option in Azure SQL Database.

## Overview

With single and pooled databases in Azure SQL Database, there are workload patterns where the allocation of underlying data files for databases can become larger than the amount of used data pages. This condition can occur when space used increases and data is subsequently deleted. The reason is because file space allocated is not automatically reclaimed when data is deleted.

Monitoring file space usage and shrinking data files may be necessary in the following scenarios:

- Allow data growth in an elastic pool when the file space allocated for its databases reaches the pool max size.
- Allow decreasing the max size of a single database or elastic pool.
- Allow changing a single database or elastic pool to a different service tier or performance tier with a lower max size.

### Monitoring file space usage

Most storage space metrics displayed in the Azure portal and the following APIs only measure the size of used data pages:

- Azure Resource Manager based metrics APIs including PowerShell [get-metrics](#)
- T-SQL: [sys.dm\\_db\\_resource\\_stats](#)

However, the following APIs also measure the size of space allocated for databases and elastic pools:

- T-SQL: [sys.resource\\_stats](#)
- T-SQL: [sys.elastic\\_pool\\_resource\\_stats](#)

### Shrinking data files

The SQL Database service does not automatically shrink data files to reclaim unused allocated space due to the potential impact to database performance. However, customers may shrink data files via self-service at a time of their choosing by following the steps described in [reclaim unused allocated space](#).

## NOTE

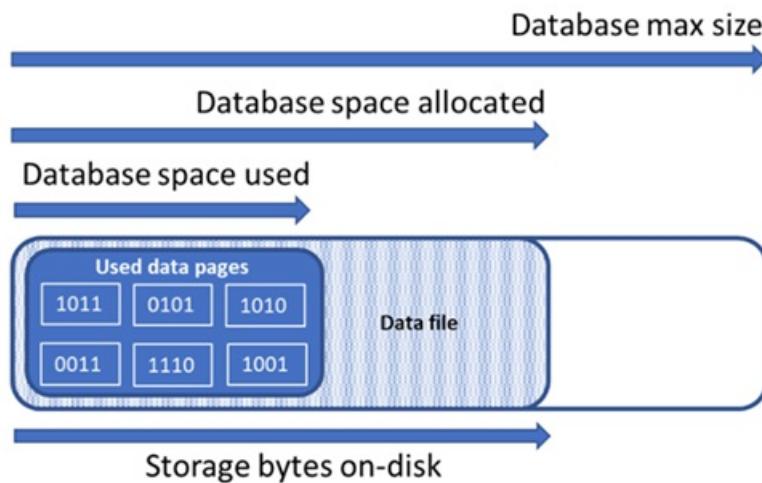
Unlike data files, the SQL Database service automatically shrinks log files since that operation does not impact database performance.

## Understanding types of storage space for a database

Understanding the following storage space quantities are important for managing the file space of a database.

DATABASE QUANTITY	DEFINITION	COMMENTS
<b>Data space used</b>	The amount of space used to store database data in 8 KB pages.	Generally, space used increases (decreases) on inserts (deletes). In some cases, the space used does not change on inserts or deletes depending on the amount and pattern of data involved in the operation and any fragmentation. For example, deleting one row from every data page does not necessarily decrease the space used.
<b>Data space allocated</b>	The amount of formatted file space made available for storing database data.	The amount of space allocated grows automatically, but never decreases after deletes. This behavior ensures that future inserts are faster since space does not need to be reformatted.
<b>Data space allocated but unused</b>	The difference between the amount of data space allocated and data space used.	This quantity represents the maximum amount of free space that can be reclaimed by shrinking database data files.
<b>Data max size</b>	The maximum amount of space that can be used for storing database data.	The amount of data space allocated cannot grow beyond the data max size.

The following diagram illustrates the relationship between the different types of storage space for a database.



## Query a single database for storage space information

The following queries can be used to determine storage space quantities for a single database.

### Database data space used

Modify the following query to return the amount of database data space used. Units of the query result are in MB.

```
-- Connect to master
-- Database data space used in MB
SELECT TOP 1 storage_in_megabytes AS DatabaseDataSpaceUsedInMB
FROM sys.resource_stats
WHERE database_name = 'db1'
ORDER BY end_time DESC
```

### Database data space allocated and unused allocated space

Use the following query to return the amount of database data space allocated and the amount of unused space allocated. Units of the query result are in MB.

```
-- Connect to database
-- Database data space allocated in MB and database data space allocated unused in MB
SELECT SUM(size/128.0) AS DatabaseDataSpaceAllocatedInMB,
SUM(size/128.0 - CAST(FILEPROPERTY(name, 'SpaceUsed') AS int)/128.0) AS
DatabaseDataSpaceAllocatedUnusedInMB
FROM sys.database_files
GROUP BY type_desc
HAVING type_desc = 'ROWS'
```

### Database data max size

Modify the following query to return the database data max size. Units of the query result are in bytes.

```
-- Connect to database
-- Database data max size in bytes
SELECT DATABASEPROPERTYEX('db1', 'MaxSizeInBytes') AS DatabaseDataMaxSizeInBytes
```

## Understanding types of storage space for an elastic pool

Understanding the following storage space quantities are important for managing the file space of an elastic pool.

ELASTIC POOL QUANTITY	DEFINITION	COMMENTS
<b>Data space used</b>	The summation of data space used by all databases in the elastic pool.	
<b>Data space allocated</b>	The summation of data space allocated by all databases in the elastic pool.	
<b>Data space allocated but unused</b>	The difference between the amount of data space allocated and data space used by all databases in the elastic pool.	This quantity represents the maximum amount of space allocated for the elastic pool that can be reclaimed by shrinking database data files.
<b>Data max size</b>	The maximum amount of data space that can be used by the elastic pool for all of its databases.	The space allocated for the elastic pool should not exceed the elastic pool max size. If this condition occurs, then space allocated that is unused can be reclaimed by shrinking database data files.

## Query an elastic pool for storage space information

The following queries can be used to determine storage space quantities for an elastic pool.

## Elastic pool data space used

Modify the following query to return the amount of elastic pool data space used. Units of the query result are in MB.

```
-- Connect to master
-- Elastic pool data space used in MB
SELECT TOP 1 avg_storage_percent / 100.0 * elastic_pool_storage_limit_mb AS ElasticPoolDataSpaceUsedInMB
FROM sys.elastic_pool_resource_stats
WHERE elastic_pool_name = 'ep1'
ORDER BY end_time DESC
```

## Elastic pool data space allocated and unused allocated space

Modify the following examples to return a table listing the space allocated and unused allocated space for each database in an elastic pool. The table orders databases from those databases with the greatest amount of unused allocated space to the least amount of unused allocated space. Units of the query result are in MB.

The query results for determining the space allocated for each database in the pool can be added together to determine the total space allocated for the elastic pool. The elastic pool space allocated should not exceed the elastic pool max size.

### IMPORTANT

The PowerShell Azure Resource Manager (RM) module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. The AzureRM module will continue to receive bug fixes until at least December 2020. The arguments for the commands in the Az module and in the AzureRm modules are substantially identical. For more about their compatibility, see [Introducing the new Azure PowerShell Az module](#).

The PowerShell script requires SQL Server PowerShell module – see [Download PowerShell module](#) to install.

```
$resourceGroupName = "<resourceGroupName>"
$serverName = "<serverName>"
$poolName = "<poolName>"
$userName = "<userName>"
$password = "<password>

get list of databases in elastic pool
$databasesInPool = Get-AzSqlElasticPoolDatabase -ResourceGroupName $resourceGroupName `
 -ServerName $serverName -ElasticPoolName $poolName
$databaseStorageMetrics = @()

for each database in the elastic pool, get space allocated in MB and space allocated unused in MB
foreach ($database in $databasesInPool) {
 $sqlCommand = "SELECT DB_NAME() as DatabaseName, `
 SUM(size/128.0) AS DatabaseDataSpaceAllocatedInMB, `
 SUM(size/128.0 - CAST(FILEPROPERTY(name, 'SpaceUsed') AS int)/128.0) AS
 DatabaseDataSpaceAllocatedUnusedInMB `
 FROM sys.database_files `
 GROUP BY type_desc `
 HAVING type_desc = 'ROWS'"
 $serverFqdn = "tcp:" + $serverName + ".database.windows.net,1433"
 $databaseStorageMetrics = $databaseStorageMetrics +
 (Invoke-Sqlcmd -ServerInstance $serverFqdn -Database $database.DatabaseName `
 -Username $userName -Password $password -Query $sqlCommand)
}

display databases in descending order of space allocated unused
Write-Output "`n" "ElasticPoolName: $poolName"
Write-Output $databaseStorageMetrics | Sort -Property DatabaseDataSpaceAllocatedUnusedInMB -Descending |
Format-Table
```

The following screenshot is an example of the output of the script:

ElasticPoolName: ep1	DatabaseName	DatabaseDataSpaceAllocatedInMB	DatabaseDataSpaceAllocatedUnusedInMB
db2		16.000000	11.062500
db3		16.000000	11.125000
db4		16.000000	11.125000
db5		16.000000	11.062500

## Elastic pool data max size

Modify the following T-SQL query to return the elastic pool data max size. Units of the query result are in MB.

```
-- Connect to master
-- Elastic pools max size in MB
SELECT TOP 1 elastic_pool_storage_limit_mb AS ElasticPoolMaxSizeInMB
FROM sys.elastic_pool_resource_stats
WHERE elastic_pool_name = 'ep1'
ORDER BY end_time DESC
```

## Reclaim unused allocated space

### NOTE

This command can impact database performance while it is running, and if possible should be run during periods of low usage.

### DBCC shrink

Once databases have been identified for reclaiming unused allocated space, modify the name of the database in the following command to shrink the data files for each database.

```
-- Shrink database data space allocated.
DBCC SHRINKDATABASE (N'db1')
```

This command can impact database performance while it is running, and if possible should be run during periods of low usage.

For more information about this command, see [SHRINKDATABASE](#).

### Auto-shrink

Alternatively, auto shrink can be enabled for a database. Auto shrink reduces file management complexity and is less impactful to database performance than `SHRINKDATABASE` or `SHRINKFILE`. Auto shrink can be particularly helpful for managing elastic pools with many databases. However, auto shrink can be less effective in reclaiming file space than `SHRINKDATABASE` and `SHRINKFILE`. To enable auto shrink, modify the name of the database in the following command.

```
-- Enable auto-shrink for the database.
ALTER DATABASE [db1] SET AUTO_SHRINK ON
```

For more information about this command, see [DATABASE SET](#) options.

### Rebuild indexes

After database data files are shrunk, indexes may become fragmented and lose their performance optimization effectiveness. If performance degradation occurs, then consider rebuilding database indexes. For more information on fragmentation and rebuilding indexes, see [Reorganize and Rebuild Indexes](#).

## Next steps

- For information about database max sizes, see:
  - [Azure SQL Database vCore-based purchasing model limits for a single database](#)
  - [Resource limits for single databases using the DTU-based purchasing model](#)
  - [Azure SQL Database vCore-based purchasing model limits for elastic pools](#)
  - [Resources limits for elastic pools using the DTU-based purchasing model](#)
- For more information about the `SHRINKDATABASE` command, see [SHRINKDATABASE](#).
- For more information on fragmentation and rebuilding indexes, see [Reorganize and Rebuild Indexes](#).

# Use Resource Health to troubleshoot connectivity for Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

## Overview

Resource Health for SQL Database helps you diagnose and get support when an Azure issue impacts your SQL resources. It informs you about the current and past health of your resources and helps you mitigate issues. Resource health provides technical support when you need help with Azure service issues.

The screenshot shows the Microsoft Azure portal interface. On the left, the sidebar includes 'Create a resource', 'Dashboard', 'All services', 'FAVORITES' (with 'All resources' selected), 'Resource groups', 'App Services', 'SQL databases', 'SQL data warehouses', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', 'Virtual networks', 'Azure Active Directory', 'Monitor', 'Advisor', 'Security Center', 'Cost Management + Billing', and 'Help + support'. The main content area is titled 'SampleDB - Resource health' and contains the following sections:

- Report a bug** and **Search (Ctrl+F)** buttons.
- Resource health** status: Shows 'Available' with the message 'There aren't any known Azure platform problems affecting this SQL database or SQL data warehouse.' and a 'Report incorrect health status' link.
- What actions can you take?** A numbered list of troubleshooting steps.
- Health history**: A table showing resource health events over the last 2 weeks, with the most recent entry being '10/28/2018 1 health event(s)'.

## Health checks

Resource Health determines the health of your SQL resource by examining the success and failure of logins to the resource. Currently, Resource Health for your SQL DB resource only examines login failures due to system error and not user error. The Resource Health status is updated every 1-2 minutes.

## Health States

### Available

A status of **Available** means that Resource Health has not detected logins failures due to system errors on your SQL resource.

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

---

## Available

There aren't any known Azure platform problems affecting this SQL database or SQL data warehouse.

[Report](#) incorrect health status

---

What actions can you take?

1. If you're having problems, use the [Troubleshooting tool](#) to get recommended solutions
  2. If your SQL database is performing slower than expected, check the Query Performance Insight tool in the [resource blade](#)
  3. If you're using geo-replication for SQL database, try [failing over to your secondary database](#)
  4. To debug slow performance on your SQL data warehouse, check the [Azure online documentation](#)
  5. If you are experiencing problems you believe are caused by Azure, [contact support](#)
- 

## Degraded

A status of **Degraded** means that Resource Health has detected a majority of successful logins, but some failures as well. These are most likely transient login errors. To reduce the impact of connection issues caused by transient login errors, please implement [retry logic](#) in your code.

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

---

### ⚠ Degraded

We're sorry your SQL database is experiencing transient login failures. Currently, Azure shows the impacted time period for your SQL database resource at a one-minute granularity. The actual impact is likely less than a minute – average is 8s.

[Report](#) incorrect health status

---

What actions can you take?

1. To reduce the impact of connection issues caused by future reconfigurations, please implement [retry logic](#) in your code.
  2. If you're having problems, use the [Troubleshooting tool](#) to get recommended solutions
  3. If you're using geo-replication for SQL database, try [failing over to your secondary database](#)
  4. If your SQL database or SQL data warehouse isn't available by the expected resolution time, [contact support](#)
- 

## Unavailable

A status of **Unavailable** means that Resource Health has detected consistent login failures to your SQL resource. If your resource remains in this state for an extended period of time, please contact support.

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

---

### ❗ Unavailable

We're sorry your SQL database is unavailable at this time. Currently, Azure shows the downtime for your SQL database resource at a one-minute granularity. The actual downtime might be less than one minute.

[Report](#) incorrect health status

---

What actions can you take?

1. If you're having problems, use the [Troubleshooting tool](#) to get recommended solutions
  2. If you're using geo-replication for SQL database, try [failing over to your secondary database](#)
  3. If your SQL database or SQL data warehouse isn't available by the expected resolution time, [contact support](#)
- 

## Unknown

The health status of **Unknown** indicates that Resource Health hasn't received information about this resource for more than 10 minutes. Although this status isn't a definitive indication of the state of the resource, it is an important data point in the troubleshooting process. If the resource is running as expected, the status of the resource will change to Available after a few minutes. If you're experiencing problems with the resource, the Unknown health status might suggest that an event in the platform is affecting the resource.

Resource health watches your resource and tells you if it's running as expected. [Learn more](#)

---

 Unknown

We're sorry we can't show the health status for your SQL database or SQL data warehouse.

[Report](#) incorrect health status

---

What actions can you take?

1. If you're having problems, use the [Troubleshooting tool](#) to get recommended solutions
  2. If your SQL database is performing slower than expected, check the Query Performance Insight tool in the [resource blade](#)
  3. If you're using geo-replication for SQL database, try [failing over to your secondary database](#) ↗
  4. To debug slow performance on your SQL data warehouse, check the [Azure online documentation](#) ↗
  5. If you are experiencing problems you believe are caused by Azure, [contact support](#)
- 

## Historical information

You can access up to 14 days of health history in the Health history section of Resource Health. The section will also contain the downtime reason (when available) for the downtimes reported by Resource Health. Currently, Azure shows the downtime for your SQL database resource at a two-minute granularity. The actual downtime is likely less than a minute – average is 8s.

### Downtime reasons

When your SQL Database experiences downtime, analysis is performed to determine a reason. When available, the downtime reason is reported in the Health History section of Resource Health. Downtime reasons are typically published 30 minutes after an event.

### Planned maintenance

The Azure infrastructure periodically performs planned maintenance – upgrade of hardware or software components in the datacenter. While the database undergoes maintenance, SQL may terminate some existing connections and refuse new ones. The login failures experienced during planned maintenance are typically transient and [retry logic](#) helps reduce the impact. If you continue to experience login errors, please contact support.

### Reconfiguration

Reconfigurations are considered transient conditions, and are expected from time to time. These events can be triggered by load balancing or software/hardware failures. Any client production application that connects to a cloud database should implement a robust connection [retry logic](#), as it would help mitigate these situations and should generally make the errors transparent to the end user.

## Next steps

- Learn more about [retry logic for transient errors](#)
- [Troubleshoot, diagnose, and prevent SQL connection errors](#)
- Learn more about [configuring Resource Health alerts](#)
- Get an overview of [Resource Health](#)
- [Resource Health FAQ](#)

# Upgrade an app to use the latest elastic database client library

11/7/2019 • 3 minutes to read • [Edit Online](#)

New versions of the [Elastic Database client library](#) are available through NuGet and the NuGet Package Manager interface in Visual Studio. Upgrades contain bug fixes and support for new capabilities of the client library.

**For the latest version:** Go to [Microsoft.Azure.SqlDatabase.ElasticScale.Client](#).

Rebuild your application with the new library, as well as change your existing Shard Map Manager metadata stored in your Azure SQL databases to support new features.

Performing these steps in order ensures that old versions of the client library are no longer present in your environment when metadata objects are updated, which means that old-version metadata objects won't be created after upgrade.

## Upgrade steps

**1. Upgrade your applications.** In Visual Studio, download and reference the latest client library version into all of your development projects that use the library; then rebuild and deploy.

- In your Visual Studio solution, select **Tools** --> **NuGet Package Manager** --> **Manage NuGet Packages for Solution**.
- (Visual Studio 2013) In the left panel, select **Updates**, and then select the **Update** button on the package **Azure SQL Database Elastic Scale Client Library** that appears in the window.
- (Visual Studio 2015) Set the Filter box to **Upgrade available**. Select the package to update, and click the **Update** button.
- (Visual Studio 2017) At the top of the dialog, select **Updates**. Select the package to update, and click the **Update** button.
- Build and Deploy.

**2. Upgrade your scripts.** If you are using **PowerShell** scripts to manage shards, [download the new library version](#) and copy it into the directory from which you execute scripts.

**3. Upgrade your split-merge service.** If you use the elastic database split-merge tool to reorganize sharded data, [download and deploy the latest version of the tool](#). Detailed upgrade steps for the Service can be found [here](#).

**4. Upgrade your Shard Map Manager databases.** Upgrade the metadata supporting your Shard Maps in Azure SQL Database. There are two ways you can accomplish this, using PowerShell or C#. Both options are shown below.

### **Option 1: Upgrade metadata using PowerShell**

1. Download the latest command-line utility for NuGet from [here](#) and save to a folder.
2. Open a Command Prompt, navigate to the same folder, and issue the command:  

```
nuget install Microsoft.Azure.SqlDatabase.ElasticScale.Client
```
3. Navigate to the subfolder containing the new client DLL version you have just downloaded, for example:  

```
cd .\Microsoft.Azure.SqlDatabase.ElasticScale.Client.1.0.0\lib\net45
```
4. Download the elastic database client upgrade script from the [Script Center](#), and save it into the same folder containing the DLL.
5. From that folder, run "PowerShell .\upgrade.ps1" from the command prompt and follow the prompts.

### **Option 2: Upgrade metadata using C#**

Alternatively, create a Visual Studio application that opens your ShardMapManager, iterates over all shards, and performs the metadata upgrade by calling the methods [UpgradeLocalStore](#) and [UpgradeGlobalStore](#) as in this example:

```
ShardMapManager smm =
 ShardMapManagerFactory.GetSqlShardMapManager(
 connStr, ShardMapManagerLoadPolicy.Lazy);
smm.UpgradeGlobalStore();

foreach (ShardLocation loc in
 smm.GetDistinctShardLocations())
{
 smm.UpgradeLocalStore(loc);
}
```

These techniques for metadata upgrades can be applied multiple times without harm. For example, if an older client version inadvertently creates a shard after you have already updated, you can run upgrade again across all shards to ensure that the latest metadata version is present throughout your infrastructure.

**Note:** New versions of the client library published to-date continue to work with prior versions of the Shard Map Manager metadata on Azure SQL DB, and vice-versa. However to take advantage of some of the new features in the latest client, metadata needs to be upgraded. Note that metadata upgrades will not affect any user-data or application-specific data, only objects created and used by the Shard Map Manager. And applications continue to operate through the upgrade sequence described above.

## Elastic database client version history

For version history, go to [Microsoft.Azure.SqlDatabase.ElasticScale.Client](#)

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Get started with Elastic Database Tools

2/4/2020 • 4 minutes to read • [Edit Online](#)

This document introduces you to the developer experience for the [elastic database client library](#) by helping you run a sample app. The sample app creates a simple sharded application and explores key capabilities of the Elastic Database Tools feature of Azure SQL Database. It focuses on use cases for [shard map management](#), [data-dependent routing](#), and [multi-shard querying](#). The client library is available for .NET as well as Java.

## Elastic Database Tools for Java

### Prerequisites

- A Java Developer Kit (JDK), version 1.8 or later
- [Maven](#)
- A SQL Database server in Azure or a local SQL Server instance

### Download and run the sample app

To build the JAR files and get started with the sample project, do the following:

1. Clone the [GitHub repository](#) containing the client library, along with the sample app.
2. Edit the `./sample/src/main/resources/resource.properties` file to set the following:
  - TEST\_CONN\_USER
  - TEST\_CONN\_PASSWORD
  - TEST\_CONN\_SERVER\_NAME
3. To build the sample project, in the `./sample` directory, run the following command:

```
mvn install
```

4. To start the sample project, in the `./sample` directory, run the following command:

```
mvn -q exec:java "-Dexec.mainClass=com.microsoft.azure.elasticdb.samples.elasticscalestarterkit.Program"
```

5. To learn more about the client library capabilities, experiment with the various options. Feel free to explore the code to learn about the sample app implementation.

```

*** Welcome to Elastic Database Tools Starter Kit ***

```

Current Range Shard Map state:

Shard Map Manager has not yet been created?

Current List Shard Map state:

Shard Map Manager has not yet been created?

1. Create shard map manager, and add a couple shards?
2. Add another shard?
3. Insert sample rows using Data-Dependent Routing?
4. Execute sample Multi-Shard Query?
5. Drop shard map manager database and all shards?
6. Exit?

Enter an option [1-6] and press ENTER:—

Congratulations! You have successfully built and run your first sharded application by using Elastic Database Tools on Azure SQL Database. Use Visual Studio or SQL Server Management Studio to connect to your SQL database and take a quick look at the shards that the sample created. You will notice new sample shard databases and a shard map manager database that the sample has created.

To add the client library to your own Maven project, add the following dependency in your POM file:

```
<dependency>
 <groupId>com.microsoft.azure</groupId>
 <artifactId>elastic-db-tools</artifactId>
 <version>1.0.0</version>
</dependency>
```

## Elastic Database Tools for .NET

### Prerequisites

- Visual Studio 2012 or later with C#. Download a free version at [Visual Studio Downloads](#).
- NuGet 2.7 or later. To get the latest version, see [Installing NuGet](#).

### Download and run the sample app

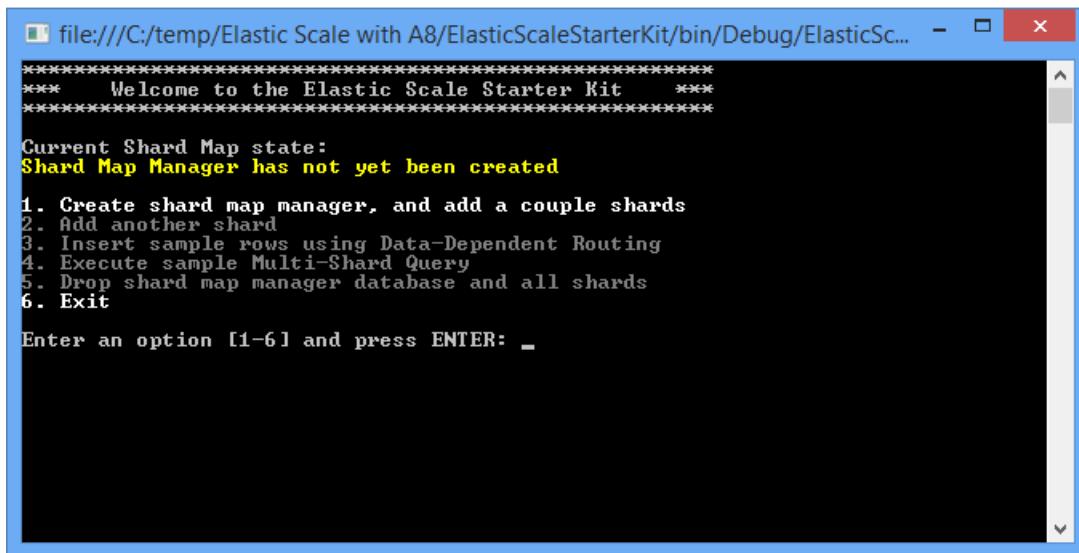
To install the library, go to [Microsoft.Azure.SqlDatabase.ElasticScale.Client](#). The library is installed with the sample app that's described in the following section.

To download and run the sample, follow these steps:

1. Download the [Elastic DB Tools for Azure SQL - Getting Started sample](#). Unzip the sample to a location that you choose.
2. To create a project, open the *ElasticScaleStarterKit.sln* solution from the C# directory.
3. In the solution for the sample project, open the *app.config* file. Then follow the instructions in the file to add your Azure SQL Database server name and your sign-in information (username and password).
4. Build and run the application. When you are prompted, enable Visual Studio to restore the NuGet

packages of the solution. This action downloads the latest version of the elastic database client library from NuGet.

5. To learn more about the client library capabilities, experiment with the various options. Note the steps that the application takes in the console output, and feel free to explore the code behind the scenes.



```
file:///C:/temp/Elastic Scale with A8/ElasticScaleStarterKit/bin/Debug/ElasticSc...

*** Welcome to the Elastic Scale Starter Kit ***

Current Shard Map state:
Shard Map Manager has not yet been created

1. Create shard map manager, and add a couple shards
2. Add another shard
3. Insert sample rows using Data-Dependent Routing
4. Execute sample Multi-Shard Query
5. Drop shard map manager database and all shards
6. Exit

Enter an option [1-6] and press ENTER: _
```

Congratulations! You have successfully built and run your first sharded application by using Elastic Database Tools on SQL Database. Use Visual Studio or SQL Server Management Studio to connect to your SQL database and take a quick look at the shards that the sample created. You will notice new sample shard databases and a shard map manager database that the sample has created.

**IMPORTANT**

We recommend that you always use the latest version of Management Studio so that you stay synchronized with updates to Azure and SQL Database. [Update SQL Server Management Studio](#).

## Key pieces of the code sample

- **Managing shards and shard maps:** The code illustrates how to work with shards, ranges, and mappings in the *ShardManagementUtils.cs* file. For more information, see [Scale out databases with the shard map manager](#).
- **Data-dependent routing:** Routing of transactions to the right shard is shown in the *DataDependentRoutingSample.cs* file. For more information, see [Data-dependent routing](#).
- **Querying over multiple shards:** Querying across shards is illustrated in the *MultiShardQuerySample.cs* file. For more information, see [Multi-shard querying](#).
- **Adding empty shards:** The iterative adding of new empty shards is performed by the code in the *CreateShardSample.cs* file. For more information, see [Scale out databases with the shard map manager](#).

## Other elastic scale operations

- **Splitting an existing shard:** The capability to split shards is provided by the split-merge tool. For more information, see [Moving data between scaled-out cloud databases](#).
- **Merging existing shards:** Shard merges are also performed by using the split-merge tool. For more information, see [Moving data between scaled-out cloud databases](#).

## Cost

The Elastic Database Tools library is free. When you use Elastic Database Tools, you incur no additional charges beyond the cost of your Azure usage.

For example, the sample application creates new databases. The cost of this capability depends on the SQL Database edition you choose and the Azure usage of your application.

For pricing information, see [SQL Database pricing details](#).

## Next steps

For more information about Elastic Database Tools, see the following articles:

- Code samples:
  - Elastic Database Tools ([.NET](#), [Java](#))
  - [Elastic Database Tools for Azure SQL - Entity Framework Integration](#)
  - [Shard Elasticity on Script Center](#)
- Blog: [Elastic Scale announcement](#)
- Channel 9: [Elastic Scale overview video](#)
- Discussion forum: [Azure SQL Database forum](#)
- To measure performance: [Performance counters for shard map manager](#)

# Report across scaled-out cloud databases (preview)

11/7/2019 • 4 minutes to read • [Edit Online](#)

You can create reports from multiple Azure SQL databases from a single connection point using an [elastic query](#). The databases must be horizontally partitioned (also known as "sharded").

If you have an existing database, see [Migrating existing databases to scaled-out databases](#).

To understand the SQL objects needed to query, see [Query across horizontally partitioned databases](#).

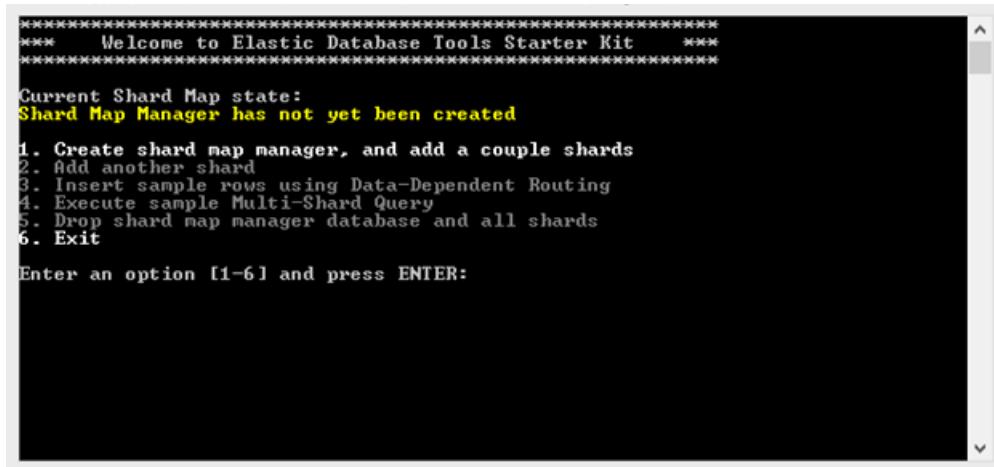
## Prerequisites

Download and run the [Getting started with Elastic Database tools sample](#).

## Create a shard map manager using the sample app

Here you will create a shard map manager along with several shards, followed by insertion of data into the shards. If you happen to already have shards setup with sharded data in them, you can skip the following steps and move to the next section.

1. Build and run the **Getting started with Elastic Database tools** sample application by following the steps in the article section [Download and run the sample app](#). Once you finish all the steps, you will see the following command prompt:



```

*** Welcome to Elastic Database Tools Starter Kit ***

Current Shard Map state:
Shard Map Manager has not yet been created
1. Create shard map manager, and add a couple shards
2. Add another shard
3. Insert sample rows using Data-Dependent Routing
4. Execute sample Multi-Shard Query
5. Drop shard map manager database and all shards
6. Exit

Enter an option [1-6] and press ENTER:
```

2. In the command window, type "1" and press **Enter**. This creates the shard map manager, and adds two shards to the server. Then type "3" and press **Enter**; repeat the action four times. This inserts sample data rows in your shards.
3. The [Azure portal](#) should show three new databases in your server:

Databases		
SQL databases		
Database	Status	Pricing Tier
ElasticScaleStarterKit_Shard1	Online	Basic
ElasticScaleStarterKit_Shard0	Online	Basic
ElasticScaleStarterKit_ShardMapManagerDb	Online	Basic

At this point, cross-database queries are supported through the Elastic Database client libraries. For example, use option 4 in the command window. The results from a multi-shard query are always a **UNION ALL** of the results from all shards.

In the next section, we create a sample database endpoint that supports richer querying of the data across shards.

# Create an elastic query database

1. Open the [Azure portal](#) and log in.
  2. Create a new Azure SQL database in the same server as your shard setup. Name the database "ElasticDBQuery."

The screenshot shows the Azure Data + Storage blade. On the left sidebar, there are icons for HOME, NOTIFICATIONS, BROWSE, ACTIVE, BILLING, and HELP. The main area displays a list of services:

- SQL Database**: Scalable and managed relational database service for modern business-class apps.
- Azure DocumentDB**: Scalable and managed NoSQL document database service for modern cloud applications.
- Storage**: Enhance existing applications with durable cloud storage, backup, and recovery.
- Redis Cache**: Distributed, in-memory Redis Cache service for modern cloud applications.
- Search**: Search-as-a-service solution.
- StorSimple**: StorSimple and Microsoft Azure offer a unique and integrated primary storage, archival, and
- MongoLab**: MongoLab is a fully-managed cloud database service featuring highly-available MongoDB
- DataStax Enterprise (production use - BYOL)**: DataStax Enterprise - The Leading Distributed Database

The central panel is titled "SQL Database" and shows the following configuration details:

- Name: ElasticDBQuery
- SERVER: [REDACTED] (North Central US)
- SELECT SOURCE: Blank database
- PRICING TIER: Standard S0
- OPTIONAL CONFIGURATION: Collation
- RESOURCE GROUP: Group-8
- SUBSCRIPTION: ElasicScaleDev\_657854

The right panel is titled "Choose your pricing tier" and lists various pricing tiers:

Choose your pricing tier		
BROWSE THE AVAILABLE PLANS AND THEIR FEATURES		
<b>P1 Premium</b>	<b>P2 Premium</b>	<b>P3 Premium</b>
100 DTUs	200 DTUs	800 DTUs
Up to 500 GB	Up to 500 GB	Up to 500 GB
Active Geo-Replicat...	Active Geo-Replicat...	Active Geo-Replicat...
Point In Time Resto...	Point In Time Resto...	Point In Time Resto...
Auditing	Auditing	Auditing
<b>465.00</b> USD/MONTH (ESTIMATED 31 P1 D...)	<b>930.00</b> USD/MONTH (ESTIMATED 31 P2 D...)	<b>3,720.00</b> USD/MONTH (ESTIMATED 31 P3 D...)
<b>S0 Standard</b>	<b>S1 Standard</b>	<b>S2 Standard</b>
10 DTUs	20 DTUs	50 DTUs
Up to 250 GB	Up to 250 GB	Up to 250 GB
Standard Geo-Replic...	Standard Geo-Replic...	Standard Geo-Replic...
Point In Time Resto...	Point In Time Resto...	Point In Time Resto...
Auditing	Auditing	Auditing
<b>15.00</b> USD/MONTH (ESTIMATED 31 S0 D...)	<b>30.00</b> USD/MONTH (ESTIMATED 31 S1 D...)	<b>75.02</b> USD/MONTH (ESTIMATED 31 S2 D...)
<b>S3 Standard</b>	<b>B Basic</b>	<b>W Web (Retired)</b>
100 DTUs	5 DTUs	- Shared Infrastructure

At the bottom, there are buttons for "Pin to Startboard" (checked) and "Create".

#### NOTE

you can use an existing database. If you can do so, it must not be one of the shards that you would like to execute your queries on. This database will be used for creating the metadata objects for an elastic database query.

## Create database objects

### Database-scoped master key and credentials

These are used to connect to the shard map manager and the shards:

1. Open SQL Server Management Studio or SQL Server Data Tools in Visual Studio.
2. Connect to ElasticDBQuery database and execute the following T-SQL commands:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<master_key_password>';

CREATE DATABASE SCOPED CREDENTIAL ElasticDBQueryCred
WITH IDENTITY = '<username>',
SECRET = '<password>';
```

"username" and "password" should be the same as login information used in step 3 of section [Download and run the sample app](#) in the **Getting started with Elastic Database tools** article.

### External data sources

To create an external data source, execute the following command on the ElasticDBQuery database:

```
CREATE EXTERNAL DATA SOURCE MyElasticDBQueryDataSrc WITH
(
 TYPE = SHARD_MAP_MANAGER,
 LOCATION = '<server_name>.database.windows.net',
 DATABASE_NAME = 'ElasticScaleStarterKit_ShardMapManagerDb',
 CREDENTIAL = ElasticDBQueryCred,
 SHARD_MAP_NAME = 'CustomerIDShardMap'
) ;
```

"CustomerIDShardMap" is the name of the shard map, if you created the shard map and shard map manager using the elastic database tools sample. However, if you used your custom setup for this sample, then it should be the shard map name you chose in your application.

### External tables

Create an external table that matches the Customers table on the shards by executing the following command on ElasticDBQuery database:

```
CREATE EXTERNAL TABLE [dbo].[Customers]
([CustomerId] [int] NOT NULL,
 [Name] [nvarchar](256) NOT NULL,
 [RegionId] [int] NOT NULL)
WITH
(DATA_SOURCE = MyElasticDBQueryDataSrc,
 DISTRIBUTION = SHARDED([CustomerId])
) ;
```

## Execute a sample elastic database T-SQL query

Once you have defined your external data source and your external tables you can now use full T-SQL over your external tables.

Execute this query on the ElasticDBQuery database:

```
select count(CustomerId) from [dbo].[Customers]
```

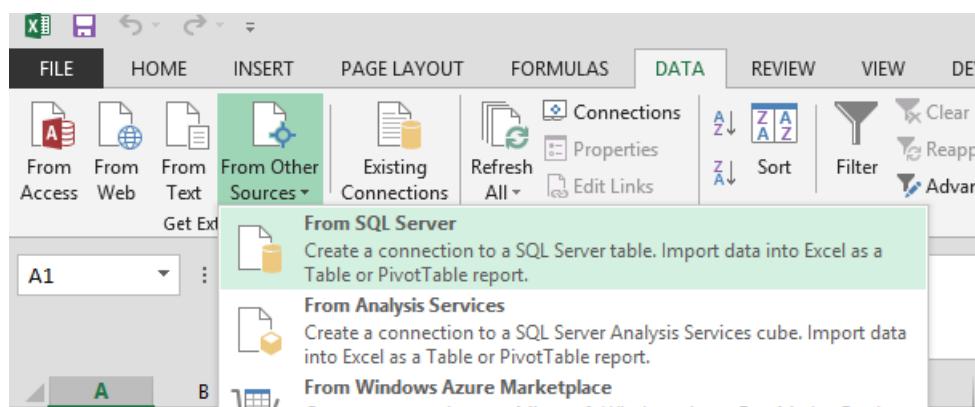
You will notice that the query aggregates results from all the shards and gives the following output:

Results	Messages
(No column name)	
1	4

## Import elastic database query results to Excel

You can import the results from of a query to an Excel file.

1. Launch Excel 2013.
2. Navigate to the **Data** ribbon.
3. Click **From Other Sources** and click **From SQL Server**.



4. In the **Data Connection Wizard** type the server name and login credentials. Then click **Next**.
5. In the dialog box **Select the database that contains the data you want**, select the **ElasticDBQuery** database.
6. Select the **Customers** table in the list view and click **Next**. Then click **Finish**.
7. In the **Import Data** form, under **Select how you want to view this data in your workbook**, select **Table** and click **OK**.

All the rows from **Customers** table, stored in different shards populate the Excel sheet.

You can now use Excel's powerful data visualization functions. You can use the connection string with your server name, database name and credentials to connect your BI and data integration tools to the elastic query database. Make sure that SQL Server is supported as a data source for your tool. You can refer to the elastic query database and external tables just like any other SQL Server database and SQL Server tables that you would connect to with your tool.

### Cost

There is no additional charge for using the Elastic Database Query feature.

For pricing information see [SQL Database Pricing Details](#).

## Next steps

- For an overview of elastic query, see [Elastic query overview](#).

- For a vertical partitioning tutorial, see [Getting started with cross-database query \(vertical partitioning\)](#).
- For syntax and sample queries for vertically partitioned data, see [Querying vertically partitioned data](#)
- For syntax and sample queries for horizontally partitioned data, see [Querying horizontally partitioned data](#)
- See [sp\\_execute\\_remote](#) for a stored procedure that executes a Transact-SQL statement on a single remote Azure SQL Database or set of databases serving as shards in a horizontal partitioning scheme.

# Multi-shard querying using elastic database tools

11/7/2019 • 2 minutes to read • [Edit Online](#)

## Overview

With the [Elastic Database tools](#), you can create sharded database solutions. **Multi-shard querying** is used for tasks such as data collection/reporting that require running a query that stretches across several shards. (Contrast this to [data-dependent routing](#), which performs all work on a single shard.)

1. Get a [RangeShardMap \(Java, .NET\)](#) or [ListShardMap \(Java, .NET\)](#) using the [TryGetRangeShardMap \(Java, .NET\)](#), the [TryGetListShardMap \(Java, .NET\)](#), or the [GetShardMap \(Java, .NET\)](#) method. See [Constructing a ShardMapManager](#) and [Get a RangeShardMap or ListShardMap](#).
2. Create a [MultiShardConnection \(Java, .NET\)](#) object.
3. Create a [MultiShardStatement](#) or [MultiShardCommand \(Java, .NET\)](#).
4. Set the [CommandText property \(Java, .NET\)](#) to a T-SQL command.
5. Execute the command by calling the [ExecuteQueryAsync](#) or [ExecuteReader \(Java, .NET\)](#) method.
6. View the results using the [MultiShardResultSet](#) or [MultiShardDataReader \(Java, .NET\)](#) class.

## Example

The following code illustrates the usage of multi-shard querying using a given **ShardMap** named *myShardMap*.

```
using (MultiShardConnection conn = new MultiShardConnection(myShardMap.GetShards(), myShardConnectionString))
{
 using (MultiShardCommand cmd = conn.CreateCommand())
 {
 cmd.CommandText = "SELECT c1, c2, c3 FROM ShardedTable";
 cmd.CommandType = CommandType.Text;
 cmd.ExecutionOptions = MultiShardExecutionOptions.IncludeShardNameColumn;
 cmd.ExecutionPolicy = MultiShardExecutionPolicy.PartialResults;

 using (MultiShardDataReader sdr = cmd.ExecuteReader())
 {
 while (sdr.Read())
 {
 var c1Field = sdr.GetString(0);
 var c2Field = sdr.GetFieldValue<int>(1);
 var c3Field = sdr.GetFieldValue<Int64>(2);
 }
 }
 }
}
```

A key difference is the construction of multi-shard connections. Where **SqlConnection** operates on an individual database, the **MultiShardConnection** takes a **collection of shards** as its input. Populate the collection of shards from a shard map. The query is then executed on the collection of shards using **UNION ALL** semantics to assemble a single overall result. Optionally, the name of the shard where the row originates from can be added to the output using the **ExecutionOptions** property on command.

Note the call to **myShardMap.GetShards()**. This method retrieves all shards from the shard map and provides an easy way to run a query across all relevant databases. The collection of shards for a multi-shard query can be refined further by performing a LINQ query over the collection returned from the call to **myShardMap.GetShards()**. In combination with the partial results policy, the current capability in multi-shard

querying has been designed to work well for tens up to hundreds of shards.

A limitation with multi-shard querying is currently the lack of validation for shards and shardlets that are queried. While data-dependent routing verifies that a given shard is part of the shard map at the time of querying, multi-shard queries do not perform this check. This can lead to multi-shard queries running on databases that have been removed from the shard map.

## Multi-shard queries and split-merge operations

Multi-shard queries do not verify whether shardlets on the queried database are participating in ongoing split-merge operations. (See [Scaling using the Elastic Database split-merge tool](#).) This can lead to inconsistencies where rows from the same shardlet show for multiple databases in the same multi-shard query. Be aware of these limitations and consider draining ongoing split-merge operations and changes to the shard map while performing multi-shard queries.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Deploy a split-merge service to move data between sharded databases

11/22/2019 • 12 minutes to read • [Edit Online](#)

The split-merge tool lets you move data between sharded databases. See [Moving data between scaled-out cloud databases](#)

## Download the Split-Merge packages

1. Download the latest NuGet version from [NuGet](#).
2. Open a command prompt and navigate to the directory where you downloaded nuget.exe. The download includes PowerShell commands.
3. Download the latest Split-Merge package into the current directory with the below command:

```
nuget install Microsoft.Azure.SqlDatabase.ElasticScale.Service.SplitMerge
```

The files are placed in a directory named

**Microsoft.Azure.SqlDatabase.ElasticScale.Service.SplitMerge.x.x.xxx.x** where x.x.xxx.x reflects the version number. Find the split-merge Service files in the **content\splitmerge\service** sub-directory, and the Split-Merge PowerShell scripts (and required client dlls) in the **content\splitmerge\powershell** sub-directory.

## Prerequisites

1. Create an Azure SQL DB database that will be used as the split-merge status database. Go to the [Azure portal](#). Create a new **SQL Database**. Give the database a name and create a new administrator and password. Be sure to record the name and password for later use.
2. Ensure that your Azure SQL DB server allows Azure Services to connect to it. In the portal, in the **Firewall Settings**, ensure the **Allow access to Azure Services** setting is set to **On**. Click the "save" icon.
3. Create an Azure Storage account for diagnostics output.
4. Create an Azure Cloud Service for your Split-Merge service.

## Configure your Split-Merge service

### Split-Merge service configuration

1. In the folder into which you downloaded the Split-Merge assemblies, create a copy of the *ServiceConfiguration.Template.cscfg* file that shipped alongside *SplitMergeService.cspkg* and rename it *ServiceConfiguration.cscfg*.
2. Open *ServiceConfiguration.cscfg* in a text editor such as Visual Studio that validates inputs such as the format of certificate thumbprints.
3. Create a new database or choose an existing database to serve as the status database for Split-Merge operations and retrieve the connection string of that database.

## IMPORTANT

At this time, the status database must use the Latin collation (SQL\_Latin1\_General\_CI\_AS). For more information, see [Windows Collation Name \(Transact-SQL\)](#).

With Azure SQL DB, the connection string typically is of the form:

```
Server=<serverName>.database.windows.net; Database=<databaseName>;User ID=<userId>; Password=<password>; Encrypt=True; Connection Timeout=30
```

4. Enter this connection string in the `.cscfg` file in both the **SplitMergeWeb** and **SplitMergeWorker** role sections in the `ElasticScaleMetadata` setting.
5. For the **SplitMergeWorker** role, enter a valid connection string to Azure storage for the **WorkerRoleSynchronizationStorageConnectionString** setting.

## Configure security

For detailed instructions to configure the security of the service, refer to the [Split-Merge security configuration](#).

For the purposes of a simple test deployment for this tutorial, a minimal set of configuration steps will be performed to get the service up and running. These steps enable only the one machine/account executing them to communicate with the service.

## Create a self-signed certificate

Create a new directory and from this directory execute the following command using a [Developer Command Prompt for Visual Studio](#) window:

```
makecert ^
-n "CN=*.cloudapp.net" ^
-r -cy end -sky exchange -eku "1.3.6.1.5.5.7.3.1,1.3.6.1.5.5.7.3.2" ^
-a sha256 -len 2048 ^
-sr currentuser -ss root ^
-sv MyCert.pvk MyCert.cer
```

You are asked for a password to protect the private key. Enter a strong password and confirm it. You are then prompted for the password to be used once more after that. Click **Yes** at the end to import it to the Trusted Certification Authorities Root store.

## Create a PFX file

Execute the following command from the same window where makecert was executed; use the same password that you used to create the certificate:

```
pvk2pfx -pvk MyCert.pvk -spc MyCert.cer -pfx MyCert.pfx -pi <password>
```

## Import the client certificate into the personal store

1. In Windows Explorer, double-click `MyCert.pfx`.
2. In the **Certificate Import Wizard** select **Current User** and click **Next**.
3. Confirm the file path and click **Next**.
4. Type the password, leave **Include all extended properties** checked and click **Next**.
5. Leave **Automatically select the certificate store[...]** checked and click **Next**.
6. Click **Finish** and **OK**.

## Upload the PFX file to the cloud service

1. Go to the [Azure portal](#).

2. Select **Cloud Services**.
3. Select the cloud service you created above for the Split/Merge service.
4. Click **Certificates** on the top menu.
5. Click **Upload** in the bottom bar.
6. Select the PFX file and enter the same password as above.
7. Once completed, copy the certificate thumbprint from the new entry in the list.

### Update the service configuration file

Paste the certificate thumbprint copied above into the thumbprint/value attribute of these settings. For the worker role:

```
<Setting name="DataEncryptionPrimaryCertificateThumbprint" value="" />
<Certificate name="DataEncryptionPrimary" thumbprint="" thumbprintAlgorithm="sha1" />
```

For the web role:

```
<Setting name="AdditionalTrustedRootCertificationAuthorities" value="" />
<Setting name="AllowedClientCertificateThumbprints" value="" />
<Setting name="DataEncryptionPrimaryCertificateThumbprint" value="" />
<Certificate name="SSL" thumbprint="" thumbprintAlgorithm="sha1" />
<Certificate name="CA" thumbprint="" thumbprintAlgorithm="sha1" />
<Certificate name="DataEncryptionPrimary" thumbprint="" thumbprintAlgorithm="sha1" />
```

Please note that for production deployments separate certificates should be used for the CA, for encryption, the Server certificate and client certificates. For detailed instructions on this, see [Security Configuration](#).

## Deploy your service

1. Go to the [Azure portal](#)
2. Select the cloud service that you created earlier.
3. Click **Overview**.
4. Choose the staging environment, then click **Upload**.
5. In the dialog box, enter a deployment label. For both 'Package' and 'Configuration', click 'From Local' and choose the *SplitMergeService.cspkg* file and your cscfg file that you configured earlier.
6. Ensure that the checkbox labeled **Deploy even if one or more roles contain a single instance** is checked.
7. Hit the tick button in the bottom right to begin the deployment. Expect it to take a few minutes to complete.

## Troubleshoot the deployment

If your web role fails to come online, it is likely a problem with the security configuration. Check that the SSL is configured as described above.

If your worker role fails to come online, but your web role succeeds, it is most likely a problem connecting to the status database that you created earlier.

- Make sure that the connection string in your cscfg is accurate.
- Check that the server and database exist, and that the user id and password are correct.
- For Azure SQL DB, the connection string should be of the form:

```
Server=<serverName>.database.windows.net; Database=<databaseName>;User ID=<user>; Password=<password>;
Encrypt=True; Connection Timeout=30
```

- Ensure that the server name does not begin with **https://**.

- Ensure that your Azure SQL DB server allows Azure Services to connect to it. To do this, open your database in the portal and ensure that the **Allow access to Azure Services** setting is set to **On**\*\*.

## Test the service deployment

### Connect with a web browser

Determine the web endpoint of your Split-Merge service. You can find this in the portal by going to the **Overview** of your cloud service and looking under **Site URL** on the right side. Replace **http://** with **https://** since the default security settings disable the HTTP endpoint. Load the page for this URL into your browser.

### Test with PowerShell scripts

The deployment and your environment can be tested by running the included sample PowerShell scripts.

The script files included are:

1. *SetupSampleSplitMergeEnvironment.ps1* - sets up a test data tier for Split/Merge (see table below for detailed description)
2. *ExecuteSampleSplitMerge.ps1* - executes test operations on the test data tier (see table below for detailed description)
3. *GetMapping.ps1* - top-level sample script that prints out the current state of the shard mappings.
4. *ShardManagement.psm1* - helper script that wraps the ShardManagement API
5. *SqlDatabaseHelpers.psm1* - helper script for creating and managing SQL databases

POWERSHELL FILE	STEPS
<b>SETUPSAMPLESPLITMERGEENVIRONMENT.PS1</b>	<ol style="list-style-type: none"> <li>1. Creates a shard map manager database</li> <li>2. Creates 2 shard databases.</li> <li>3. Creates a shard map for those databases (deletes any existing shard maps on those databases).</li> <li>4. Creates a small sample table in both the shards, and populates the table in one of the shards.</li> <li>5. Declares the SchemaInfo for the sharded table.</li> </ol>
<b>EXECUTESAMPLESPLITMERGE.PS1</b>	<ol style="list-style-type: none"> <li>1. Sends a split request to the Split-Merge Service web frontend, which splits half the data from the first shard to the second shard.</li> <li>2. Polls the web frontend for the split request status and waits until the request completes.</li> <li>3. Sends a merge request to the Split-Merge Service web frontend, which moves the data from the second shard back to the first shard.</li> <li>4. Polls the web frontend for the merge request status and waits until the request completes.</li> </ol>

# Use PowerShell to verify your deployment

1. Open a new PowerShell window and navigate to the directory where you downloaded the Split-Merge package, and then navigate into the "powershell" directory.
2. Create an Azure SQL Database server (or choose an existing server) where the shard map manager and shards will be created.

## NOTE

The *SetupSampleSplitMergeEnvironment.ps1* script creates all these databases on the same server by default to keep the script simple. This is not a restriction of the Split-Merge Service itself.

A SQL authentication login with read/write access to the DBs will be needed for the Split-Merge service to move data and update the shard map. Since the Split-Merge Service runs in the cloud, it does not currently support Integrated Authentication.

Make sure the Azure SQL server is configured to allow access from the IP address of the machine running these scripts. You can find this setting under the Azure SQL server / configuration / allowed IP addresses.

3. Execute the *SetupSampleSplitMergeEnvironment.ps1* script to create the sample environment.

Running this script will wipe out any existing shard map management data structures on the shard map manager database and the shards. It may be useful to rerun the script if you wish to re-initialize the shard map or shards.

Sample command line:

```
.\SetupSampleSplitMergeEnvironment.ps1
-UserName 'mysqluser' -Password 'MySqlPassw0rd' -ShardMapManagerServerName
'abcdefghijkl.database.windows.net'
```

4. Execute the *GetMappings.ps1* script to view the mappings that currently exist in the sample environment.

```
.\GetMappings.ps1
-UserName 'mysqluser' -Password 'MySqlPassw0rd' -ShardMapManagerServerName
'abcdefghijkl.database.windows.net'
```

5. Execute the *ExecuteSampleSplitMerge.ps1* script to execute a split operation (moving half the data on the first shard to the second shard) and then a merge operation (moving the data back onto the first shard). If you configured SSL and left the http endpoint disabled, ensure that you use the https:// endpoint instead.

Sample command line:

```
.\ExecuteSampleSplitMerge.ps1
-UserName 'mysqluser' -Password 'MySqlPassw0rd'
-ShardMapManagerServerName 'abcdefghijkl.database.windows.net'
-SplitMergeServiceEndpoint 'https://mysplitmergeservice.cloudapp.net'
-CertificateThumbprint '0123456789abcdef0123456789abcdef01234567'
```

If you receive the below error, it is most likely a problem with your Web endpoint's certificate. Try connecting to the Web endpoint with your favorite Web browser and check if there is a certificate error.

```
Invoke-WebRequest : The underlying connection was closed: Could not establish trust relationship for the
SSL/TLSsecure channel.
```

If it succeeded, the output should look like the below:

```

> .\ExecuteSampleSplitMerge.ps1 -UserName 'mysqluser' -Password 'MySqlPassw0rd' -
ShardMapManagerServerName 'abcdefghijkl.database.windows.net' -SplitMergeServiceEndpoint
'http://mysplitmergeservice.cloudapp.net/' -CertificateThumbprint
0123456789abcdef0123456789abcdef01234567
> Sending split request
> Began split operation with id dc68dfa0-e22b-4823-886a-9bdc903c80f3
> Polling split-merge request status. Press Ctrl-C to end
> Progress: 0% | Status: Queued | Details: [Informational] Queued request
> Progress: 5% | Status: Starting | Details: [Informational] Starting split-merge state machine for
request.
> Progress: 5% | Status: Starting | Details: [Informational] Performing data consistency checks on
target shards.
> Progress: 20% | Status: CopyingReferenceTables | Details: [Informational] Moving reference tables
from source to target shard.
> Progress: 20% | Status: CopyingReferenceTables | Details: [Informational] Waiting for reference
tables copy completion.
> Progress: 20% | Status: CopyingReferenceTables | Details: [Informational] Moving reference tables
from source to target shard.
> Progress: 44% | Status: CopyingShardedTables | Details: [Informational] Moving key range [100:110) of
Sharded tables
> Progress: 44% | Status: CopyingShardedTables | Details: [Informational] Successfully copied key range
[100:110) for table [dbo].[MyShardedTable]
> ...
> ...
> Progress: 90% | Status: Completing | Details: [Informational] Successfully deleted shardlets in table
[dbo].[MyShardedTable].
> Progress: 90% | Status: Completing | Details: [Informational] Deleting any temp tables that were
created while processing the request.
> Progress: 100% | Status: Succeeded | Details: [Informational] Successfully processed request.
> Sending merge request
> Began merge operation with id 6ffc308f-d006-466b-b24e-857242ec5f66
> Polling request status. Press Ctrl-C to end
> Progress: 0% | Status: Queued | Details: [Informational] Queued request
> Progress: 5% | Status: Starting | Details: [Informational] Starting split-merge state machine for
request.
> Progress: 5% | Status: Starting | Details: [Informational] Performing data consistency checks on
target shards.
> Progress: 20% | Status: CopyingReferenceTables | Details: [Informational] Moving reference tables
from source to target shard.
> Progress: 44% | Status: CopyingShardedTables | Details: [Informational] Moving key range [100:110) of
Sharded tables
> Progress: 44% | Status: CopyingShardedTables | Details: [Informational] Successfully copied key range
[100:110) for table [dbo].[MyShardedTable]
> ...
> ...
> Progress: 90% | Status: Completing | Details: [Informational] Successfully deleted shardlets in table
[dbo].[MyShardedTable].
> Progress: 90% | Status: Completing | Details: [Informational] Deleting any temp tables that were
created while processing the request.
> Progress: 100% | Status: Succeeded | Details: [Informational] Successfully processed request.
>

```

6. Experiment with other data types! All of these scripts take an optional -ShardKeyType parameter that allows you to specify the key type. The default is Int32, but you can also specify Int64, Guid, or Binary.

## Create requests

The service can be used either by using the web UI or by importing and using the SplitMerge.psm1 PowerShell module which will submit your requests through the web role.

The service can move data in both sharded tables and reference tables. A sharded table has a sharding key column and has different row data on each shard. A reference table is not sharded so it contains the same row data on every shard. Reference tables are useful for data that does not change often and is used to JOIN with sharded tables in queries.

In order to perform a split-merge operation, you must declare the sharded tables and reference tables that you want to have moved. This is accomplished with the **SchemaInfo** API. This API is in the **Microsoft.Azure.SqlDatabase.ElasticScale.ShardManagement.Schema** namespace.

1. For each sharded table, create a **ShardedTableInfo** object describing the table's parent schema name (optional, defaults to "dbo"), the table name, and the column name in that table that contains the sharding key.
2. For each reference table, create a **ReferenceTableInfo** object describing the table's parent schema name (optional, defaults to "dbo") and the table name.
3. Add the above TableInfo objects to a new **SchemaInfo** object.
4. Get a reference to a **ShardMapManager** object, and call **GetSchemaInfoCollection**.
5. Add the **SchemaInfo** to the **SchemaInfoCollection**, providing the shard map name.

An example of this can be seen in the `SetupSampleSplitMergeEnvironment.ps1` script.

The Split-Merge service does not create the target database (or schema for any tables in the database) for you. They must be pre-created before sending a request to the service.

## Troubleshooting

You may see the below message when running the sample powershell scripts:

```
Invoke-WebRequest : The underlying connection was closed: Could not establish trust relationship for the
SSL/TLS secure channel.
```

This error means that your SSL certificate is not configured correctly. Please follow the instructions in section 'Connecting with a web browser'.

If you cannot submit requests you may see this:

```
[Exception] System.Data.SqlClient.SqlException (0x80131904): Could not find stored procedure
'dbo.InsertRequest'.
```

In this case, check your configuration file, in particular the setting for **WorkerRoleSynchronizationStorageAccountConnectionString**. This error typically indicates that the worker role could not successfully initialize the metadata database on first use.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Split-merge security configuration

11/7/2019 • 12 minutes to read • [Edit Online](#)

To use the Split/Merge service, you must correctly configure security. The service is part of the Elastic Scale feature of Microsoft Azure SQL Database. For more information, see [Elastic Scale Split and Merge Service Tutorial](#).

## Configuring certificates

Certificates are configured in two ways.

1. [To Configure the SSL Certificate](#)
2. [To Configure Client Certificates](#)

## To obtain certificates

Certificates can be obtained from public Certificate Authorities (CAs) or from the [Windows Certificate Service](#). These are the preferred methods to obtain certificates.

If those options are not available, you can generate **self-signed certificates**.

## Tools to generate certificates

- [makecert.exe](#)
- [pvk2pfx.exe](#)

### To run the tools

- From a Developer Command Prompt for Visual Studios, see [Visual Studio Command Prompt](#)

If installed, go to:

```
%ProgramFiles(x86)%\Windows Kits\x.y\bin\x86
```

- Get the WDK from [Windows 8.1: Download kits and tools](#)

## To configure the SSL certificate

An SSL certificate is required to encrypt the communication and authenticate the server. Choose the most applicable of the three scenarios below, and execute all its steps:

### Create a new self-signed certificate

1. [Create a Self-Signed Certificate](#)
2. [Create PFX file for Self-Signed SSL Certificate](#)
3. [Upload SSL Certificate to Cloud Service](#)
4. [Update SSL Certificate in Service Configuration File](#)
5. [Import SSL Certification Authority](#)

### To use an existing certificate from the certificate store

1. [Export SSL Certificate From Certificate Store](#)
2. [Upload SSL Certificate to Cloud Service](#)

### [3. Update SSL Certificate in Service Configuration File](#)

#### **To use an existing certificate in a PFX file**

- [1. Upload SSL Certificate to Cloud Service](#)
- [2. Update SSL Certificate in Service Configuration File](#)

## To configure client certificates

Client certificates are required in order to authenticate requests to the service. Choose the most applicable of the three scenarios below, and execute all its steps:

#### **Turn off client certificates**

- [1. Turn Off Client Certificate-Based Authentication](#)

#### **Issue new self-signed client certificates**

- [1. Create a Self-Signed Certification Authority](#)
- [2. Upload CA Certificate to Cloud Service](#)
- [3. Update CA Certificate in Service Configuration File](#)
- [4. Issue Client Certificates](#)
- [5. Create PFX files for Client Certificates](#)
- [6. Import Client Certificate](#)
- [7. Copy Client Certificate Thumbprints](#)
- [8. Configure Allowed Clients in the Service Configuration File](#)

#### **Use existing client certificates**

- [1. Find CA Public Key](#)
- [2. Upload CA Certificate to Cloud Service](#)
- [3. Update CA Certificate in Service Configuration File](#)
- [4. Copy Client Certificate Thumbprints](#)
- [5. Configure Allowed Clients in the Service Configuration File](#)
- [6. Configure Client Certificate Revocation Check](#)

## Allowed IP addresses

Access to the service endpoints can be restricted to specific ranges of IP addresses.

## To configure encryption for the store

A certificate is required to encrypt the credentials that are stored in the metadata store. Choose the most applicable of the three scenarios below, and execute all its steps:

#### **Use a new self-signed certificate**

- [1. Create a Self-Signed Certificate](#)
- [2. Create PFX file for Self-Signed Encryption Certificate](#)
- [3. Upload Encryption Certificate to Cloud Service](#)
- [4. Update Encryption Certificate in Service Configuration File](#)

#### **Use an existing certificate from the certificate store**

- [1. Export Encryption Certificate From Certificate Store](#)
- [2. Upload Encryption Certificate to Cloud Service](#)
- [3. Update Encryption Certificate in Service Configuration File](#)

## Use an existing certificate in a PFX file

1. [Upload Encryption Certificate to Cloud Service](#)
2. [Update Encryption Certificate in Service Configuration File](#)

## The default configuration

The default configuration denies all access to the HTTP endpoint. This is the recommended setting, since the requests to these endpoints may carry sensitive information like database credentials. The default configuration allows all access to the HTTPS endpoint. This setting may be restricted further.

### Changing the Configuration

The group of access control rules that apply to an endpoint are configured in the **<EndpointAcls>** section in the **service configuration file**.

```
<EndpointAcls>
 <EndpointAcl role="SplitMergeWeb" endPoint="HttpIn" accessControl="DenyAll" />
 <EndpointAcl role="SplitMergeWeb" endPoint="HttpsIn" accessControl="AllowAll" />
</EndpointAcls>
```

The rules in an access control group are configured in a **<AccessControl name="">** section of the service configuration file.

The format is explained in Network Access Control Lists documentation. For example, to allow only IPs in the range 100.100.0.0 to 100.100.255.255 to access the HTTPS endpoint, the rules would look like this:

```
<AccessControl name="Restricted">
 <Rule action="permit" description="Some" order="1" remoteSubnet="100.100.0.0/16"/>
 <Rule action="deny" description="None" order="2" remoteSubnet="0.0.0.0/0" />
</AccessControl>
<EndpointAcls>
 <EndpointAcl role="SplitMergeWeb" endPoint="HttpsIn" accessControl="Restricted" />
</EndpointAcls>
```

## Denial of service prevention

There are two different mechanisms supported to detect and prevent Denial of Service attacks:

- Restrict number of concurrent requests per remote host (off by default)
- Restrict rate of access per remote host (on by default)

These are based on the features further documented in Dynamic IP Security in IIS. When changing this configuration beware of the following factors:

- The behavior of proxies and Network Address Translation devices over the remote host information
- Each request to any resource in the web role is considered (for example, loading scripts, images, etc)

## Restricting number of concurrent accesses

The settings that configure this behavior are:

```
<Setting name="DynamicIpRestrictionDenyByConcurrentRequests" value="false" />
<Setting name="DynamicIpRestrictionMaxConcurrentRequests" value="20" />
```

Change `DynamicIpRestrictionDenyByConcurrentRequests` to true to enable this protection.

## Restricting rate of access

The settings that configure this behavior are:

```
<Setting name="DynamicIpRestrictionDenyByRequestRate" value="true" />
<Setting name="DynamicIpRestrictionMaxRequests" value="100" />
<Setting name="DynamicIpRestrictionRequestIntervalInMilliseconds" value="2000" />
```

## Configuring the response to a denied request

The following setting configures the response to a denied request:

```
<Setting name="DynamicIpRestrictionDenyAction" value="AbortRequest" />
```

Refer to the documentation for Dynamic IP Security in IIS for other supported values.

## Operations for configuring service certificates

This topic is for reference only. Follow the configuration steps outlined in:

- Configure the SSL certificate
- Configure client certificates

### Create a self-signed certificate

Execute:

```
makecert ^
-n "CN=myservice.cloudapp.net" ^
-e MM/DD/YYYY ^
-r -cy end -sky exchange -eku "1.3.6.1.5.5.7.3.1" ^
-a sha256 -len 2048 ^
-sv MySSL.pvk MySSL.cer
```

To customize:

- -n with the service URL. Wildcards ("CN=\*.cloudapp.net") and alternative names ("CN=myservice1.cloudapp.net, CN=myservice2.cloudapp.net") are supported.
- -e with the certificate expiration date Create a strong password and specify it when prompted.

### Create PFX file for self-signed SSL certificate

Execute:

```
pvk2pfx -pvk MySSL.pvk -spc MySSL.cer
```

Enter password and then export certificate with these options:

- Yes, export the private key
- Export all extended properties

### Export SSL certificate from certificate store

- Find certificate

- Click Actions -> All tasks -> Export...
- Export certificate into a .PFX file with these options:
  - Yes, export the private key
  - Include all certificates in the certification path if possible \*Export all extended properties

## Upload SSL certificate to cloud service

Upload certificate with the existing or generated .PFX file with the SSL key pair:

- Enter the password protecting the private key information

## Update SSL certificate in service configuration file

Update the thumbprint value of the following setting in the service configuration file with the thumbprint of the certificate uploaded to the cloud service:

```
<Certificate name="SSL" thumbprint="" thumbprintAlgorithm="sha1" />
```

## Import SSL certification authority

Follow these steps in all account/machine that will communicate with the service:

- Double-click the .CER file in Windows Explorer
- In the Certificate dialog, click Install Certificate...
- Import certificate into the Trusted Root Certification Authorities store

## Turn off client certificate-based authentication

Only client certificate-based authentication is supported and disabling it will allow for public access to the service endpoints, unless other mechanisms are in place (for example, Microsoft Azure Virtual Network).

Change these settings to false in the service configuration file to turn off the feature:

```
<Setting name="SetupWebAppForClientCertificates" value="false" />
<Setting name="SetupWebserverForClientCertificates" value="false" />
```

Then, copy the same thumbprint as the SSL certificate in the CA certificate setting:

```
<Certificate name="CA" thumbprint="" thumbprintAlgorithm="sha1" />
```

## Create a self-signed certification authority

Execute the following steps to create a self-signed certificate to act as a Certification Authority:

```
makecert ^
-n "CN=MyCA" ^
-e MM/DD/YYYY ^
-r -cy authority -h 1 ^
-a sha256 -len 2048 ^
-sr localmachine -ss my ^
MyCA.cer
```

To customize it

- -e with the certification expiration date

## Find CA public key

All client certificates must have been issued by a Certification Authority trusted by the service. Find the public key to the Certification Authority that issued the client certificates that are going to be used for authentication in order to upload it to the cloud service.

If the file with the public key is not available, export it from the certificate store:

- Find certificate
  - Search for a client certificate issued by the same Certification Authority
- Double-click the certificate.
- Select the Certification Path tab in the Certificate dialog.
- Double-click the CA entry in the path.
- Take notes of the certificate properties.
- Close the **Certificate** dialog.
- Find certificate
  - Search for the CA noted above.
- Click Actions -> All tasks -> Export...
- Export certificate into a .CER with these options:
  - **No, do not export the private key**
  - Include all certificates in the certification path if possible.
  - Export all extended properties.

## Upload CA certificate to cloud service

Upload certificate with the existing or generated .CER file with the CA public key.

## Update CA certificate in service configuration file

Update the thumbprint value of the following setting in the service configuration file with the thumbprint of the certificate uploaded to the cloud service:

```
<Certificate name="CA" thumbprint="" thumbprintAlgorithm="sha1" />
```

Update the value of the following setting with the same thumbprint:

```
<Setting name="AdditionalTrustedRootCertificationAuthorities" value="" />
```

## Issue client certificates

Each individual authorized to access the service should have a client certificate issued for their exclusive use and should choose their own strong password to protect its private key.

The following steps must be executed in the same machine where the self-signed CA certificate was generated and stored:

```
makecert ^
-n "CN=My ID" ^
-e MM/DD/YYYY ^
-cy end -sky exchange -eku "1.3.6.1.5.5.7.3.2" ^
-a sha256 -len 2048 ^
-in "MyCA" -ir localmachine -is my ^
-sv MyID.pvk MyID.cer
```

Customizing:

- -n with an ID for the client that will be authenticated with this certificate
- -e with the certificate expiration date
- MyID.pvk and MyID.cer with unique filenames for this client certificate

This command will prompt for a password to be created and then used once. Use a strong password.

## Create PFX files for client certificates

For each generated client certificate, execute:

```
pvk2pfx -pvk MyID.pvk -spc MyID.cer
```

Customizing:

```
MyID.pvk and MyID.cer with the filename for the client certificate
```

Enter password and then export certificate with these options:

- Yes, export the private key
- Export all extended properties
- The individual to whom this certificate is being issued should choose the export password

## Import client certificate

Each individual for whom a client certificate has been issued should import the key pair in the machines they will use to communicate with the service:

- Double-click the .PFX file in Windows Explorer
- Import certificate into the Personal store with at least this option:
  - Include all extended properties checked

## Copy client certificate thumbprints

Each individual for whom a client certificate has been issued must follow these steps in order to obtain the thumbprint of their certificate, which will be added to the service configuration file:

- Run certmgr.exe
- Select the Personal tab
- Double-click the client certificate to be used for authentication
- In the Certificate dialog that opens, select the Details tab
- Make sure Show is displaying All
- Select the field named Thumbprint in the list
- Copy the value of the thumbprint

- Delete non-visible Unicode characters in front of the first digit
- Delete all spaces

## Configure Allowed clients in the service configuration file

Update the value of the following setting in the service configuration file with a comma-separated list of the thumbprints of the client certificates allowed access to the service:

```
<Setting name="AllowedClientCertificateThumbprints" value="" />
```

## Configure client certificate revocation check

The default setting does not check with the Certification Authority for client certificate revocation status. To turn on the checks, if the Certification Authority that issued the client certificates supports such checks, change the following setting with one of the values defined in the X509RevocationMode Enumeration:

```
<Setting name="ClientCertificateRevocationCheck" value="NoCheck" />
```

## Create PFX file for self-signed encryption certificates

For an encryption certificate, execute:

```
pvk2pfx -pvk MyID.pvk -spc MyID.cer
```

Customizing:

MyID.pvk and MyID.cer with the filename for the encryption certificate

Enter password and then export certificate with these options:

- Yes, export the private key
- Export all extended properties
- You will need the password when uploading the certificate to the cloud service.

## Export encryption certificate from certificate store

- Find certificate
- Click Actions -> All tasks -> Export...
- Export certificate into a .PFX file with these options:
  - Yes, export the private key
  - Include all certificates in the certification path if possible
- Export all extended properties

## Upload encryption certificate to cloud service

Upload certificate with the existing or generated .PFX file with the encryption key pair:

- Enter the password protecting the private key information

## Update encryption certificate in service configuration file

Update the thumbprint value of the following settings in the service configuration file with the thumbprint of the certificate uploaded to the cloud service:

```
<Certificate name="DataEncryptionPrimary" thumbprint="" thumbprintAlgorithm="sha1" />
```

## Common certificate operations

- Configure the SSL certificate
- Configure client certificates

## Find certificate

Follow these steps:

1. Run mmc.exe.
2. File -> Add/Remove Snap-in...
3. Select **Certificates**.
4. Click **Add**.
5. Choose the certificate store location.
6. Click **Finish**.
7. Click **OK**.
8. Expand **Certificates**.
9. Expand the certificate store node.
10. Expand the Certificate child node.
11. Select a certificate in the list.

## Export certificate

In the **Certificate Export Wizard**:

1. Click **Next**.
2. Select **Yes**, then **Export the private key**.
3. Click **Next**.
4. Select the desired output file format.
5. Check the desired options.
6. Check **Password**.
7. Enter a strong password and confirm it.
8. Click **Next**.
9. Type or browse a filename where to store the certificate (use a .PFX extension).
10. Click **Next**.
11. Click **Finish**.
12. Click **OK**.

## Import certificate

In the Certificate Import Wizard:

1. Select the store location.
  - Select **Current User** if only processes running under current user will access the service
  - Select **Local Machine** if other processes in this computer will access the service

2. Click **Next**.
3. If importing from a file, confirm the file path.
4. If importing a .PFX file:
  - a. Enter the password protecting the private key
  - b. Select import options
5. Select "Place" certificates in the following store
6. Click **Browse**.
7. Select the desired store.
8. Click **Finish**.
  - If the Trusted Root Certification Authority store was chosen, click **Yes**.
9. Click **OK** on all dialog windows.

## Upload certificate

In the [Azure portal](#)

1. Select **Cloud Services**.
2. Select the cloud service.
3. On the top menu, click **Certificates**.
4. On the bottom bar, click **Upload**.
5. Select the certificate file.
6. If it is a .PFX file, enter the password for the private key.
7. Once completed, copy the certificate thumbprint from the new entry in the list.

## Other security considerations

The SSL settings described in this document encrypt communication between the service and its clients when the HTTPS endpoint is used. This is important since credentials for database access and potentially other sensitive information are contained in the communication. Note, however, that the service persists internal status, including credentials, in its internal tables in the Microsoft Azure SQL database that you have provided for metadata storage in your Microsoft Azure subscription. That database was defined as part of the following setting in your service configuration file (.CSCFG file):

```
<Setting name="ElasticScaleMetadata" value="Server=..." />
```

Credentials stored in this database are encrypted. However, as a best practice, ensure that both web and worker roles of your service deployments are kept up to date and secure as they both have access to the metadata database and the certificate used for encryption and decryption of stored credentials.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Adding a shard using Elastic Database tools

11/7/2019 • 2 minutes to read • [Edit Online](#)

## To add a shard for a new range or key

Applications often need to add new shards to handle data that is expected from new keys or key ranges, for a shard map that already exists. For example, an application sharded by Tenant ID may need to provision a new shard for a new tenant, or data sharded monthly may need a new shard provisioned before the start of each new month.

If the new range of key values is not already part of an existing mapping, it is simple to add the new shard and associate the new key or range to that shard.

### Example: adding a shard and its range to an existing shard map

This sample uses the TryGetShard ([Java](#), [.NET](#)) the CreateShard ([Java](#), [.NET](#)), CreateRangeMapping ([Java](#), [.NET](#)) methods, and creates an instance of the ShardLocation ([Java](#), [.NET](#)) class. In the sample below, a database named **sample\_shard\_2** and all necessary schema objects inside of it have been created to hold range [300, 400).

```
// sm is a RangeShardMap object.
// Add a new shard to hold the range being added.
Shard shard2 = null;

if (!sm.TryGetShard(new ShardLocation(shardServer, "sample_shard_2"),out shard2))
{
 shard2 = sm.CreateShard(new ShardLocation(shardServer, "sample_shard_2"));
}

// Create the mapping and associate it with the new shard
sm.CreateRangeMapping(new RangeMappingCreateInfo<long>
 (new Range<long>(300, 400), shard2, MappingStatus.Online));
```

For the .NET version, you can also use PowerShell as an alternative to create a new Shard Map Manager. An example is available [here](#).

## To add a shard for an empty part of an existing range

In some circumstances, you may have already mapped a range to a shard and partially filled it with data, but you now want upcoming data to be directed to a different shard. For example, you shard by day range and have already allocated 50 days to a shard, but on day 24, you want future data to land in a different shard. The elastic database [split-merge tool](#) can perform this operation, but if data movement is not necessary (for example, data for the range of days [25, 50), that is, day 25 inclusive to 50 exclusive, does not yet exist) you can perform this entirely using the Shard Map Management APIs directly.

### Example: splitting a range and assigning the empty portion to a newly added shard

A database named "sample\_shard\_2" and all necessary schema objects inside of it have been created.

```
// sm is a RangeShardMap object.
// Add a new shard to hold the range we will move
Shard shard2 = null;

if (!sm.TryGetShard(new ShardLocation(shardServer, "sample_shard_2"),out shard2))
{
 shard2 = sm.CreateShard(new ShardLocation(shardServer,"sample_shard_2"));
}

// Split the Range holding Key 25
sm.SplitMapping(sm.GetMappingForKey(25), 25);

// Map new range holding [25-50) to different shard:
// first take existing mapping offline
sm.MarkMappingOffline(sm.GetMappingForKey(25));

// now map while offline to a different shard and take online
RangeMappingUpdate upd = new RangeMappingUpdate();
upd.Shard = shard2;
sm.MarkMappingOnline(sm.UpdateMapping(sm.GetMappingForKey(25), upd));
```

**Important:** Use this technique only if you are certain that the range for the updated mapping is empty. The preceding methods do not check data for the range being moved, so it is best to include checks in your code. If rows exist in the range being moved, the actual data distribution will not match the updated shard map. Use the [split-merge tool](#) to perform the operation instead in these cases.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

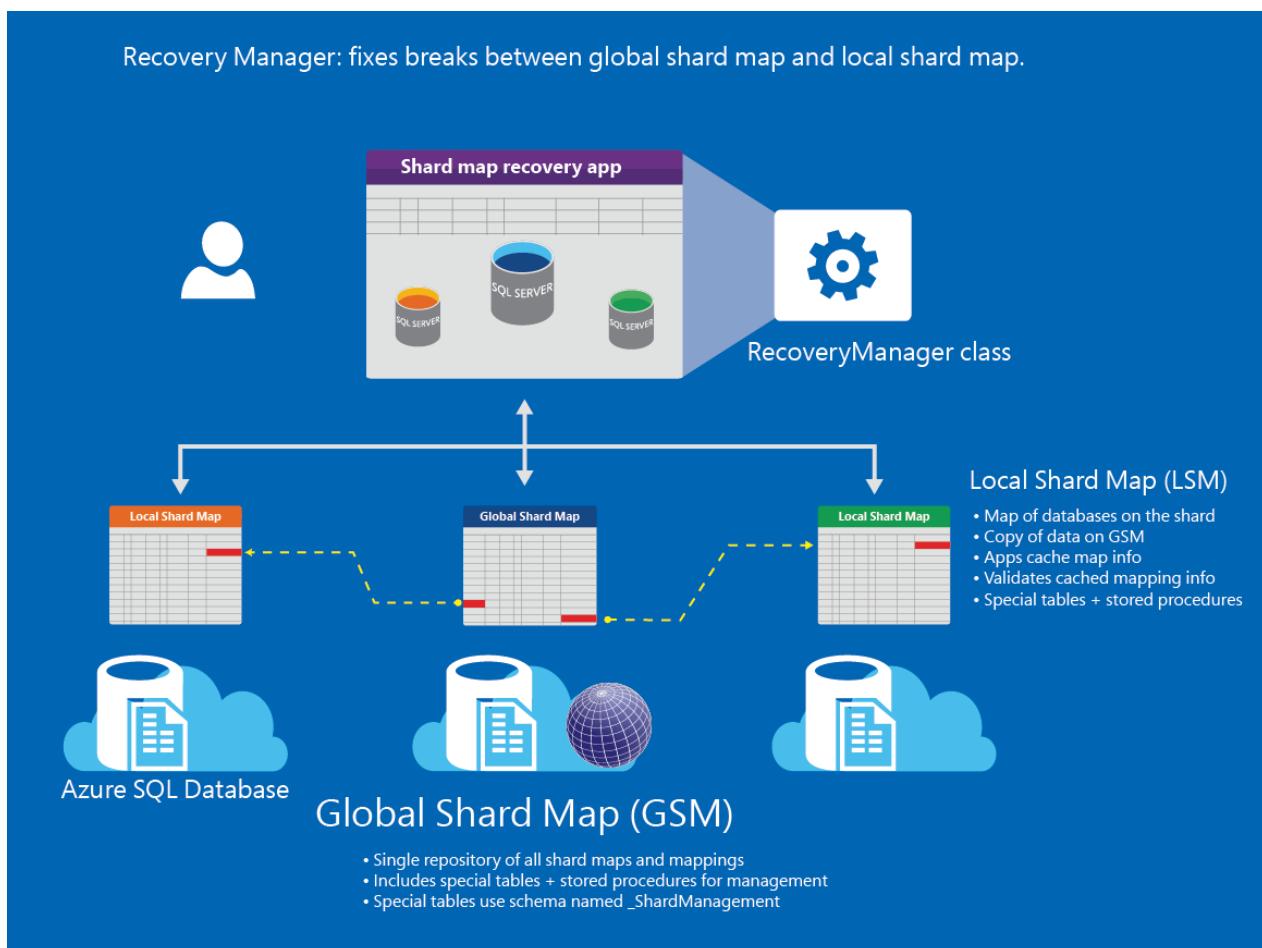
# Using the RecoveryManager class to fix shard map problems

11/7/2019 • 8 minutes to read • [Edit Online](#)

The [RecoveryManager](#) class provides ADO.NET applications the ability to easily detect and correct any inconsistencies between the global shard map (GSM) and the local shard map (LSM) in a sharded database environment.

The GSM and LSM track the mapping of each database in a sharded environment. Occasionally, a break occurs between the GSM and the LSM. In that case, use the RecoveryManager class to detect and repair the break.

The RecoveryManager class is part of the [Elastic Database client library](#).



For term definitions, see [Elastic Database tools glossary](#). To understand how the **ShardMapManager** is used to manage data in a sharded solution, see [Shard map management](#).

## Why use the recovery manager

In a sharded database environment, there is one tenant per database, and many databases per server. There can also be many servers in the environment. Each database is mapped in the shard map, so calls can be routed to the correct server and database. Databases are tracked according to a **sharding key**, and each shard is assigned a **range of key values**. For example, a sharding key may represent the customer names from "D" to "F." The mapping of all shards (aka databases) and their mapping ranges are contained in the **global shard map (GSM)**. Each database also contains a map of the ranges contained on the shard that is known as the **local shard map (LSM)**. When an app connects to a shard, the mapping is cached with the app for quick retrieval. The LSM is used

to validate cached data.

The GSM and LSM may become out of sync for the following reasons:

1. The deletion of a shard whose range is believed to no longer be in use, or renaming of a shard. Deleting a shard results in an **orphaned shard mapping**. Similarly, a renamed database can cause an orphaned shard mapping. Depending on the intent of the change, the shard may need to be removed or the shard location needs to be updated. To recover a deleted database, see [Restore a deleted database](#).
2. A geo-failover event occurs. To continue, one must update the server name, and database name of shard map manager in the application and then update the shard mapping details for all shards in a shard map. If there is a geo-failover, such recovery logic should be automated within the failover workflow. Automating recovery actions enables a frictionless manageability for geo-enabled databases and avoids manual human actions. To learn about options to recover a database if there is a data center outage, see [Business Continuity](#) and [Disaster Recovery](#).
3. Either a shard or the ShardMapManager database is restored to an earlier point-in time. To learn about point in time recovery using backups, see [Recovery using backups](#).

For more information about Azure SQL Database Elastic Database tools, geo-replication and Restore, see the following:

- [Overview: Cloud business continuity and database disaster recovery with SQL Database](#)
- [Get started with elastic database tools](#)
- [ShardMap Management](#)

## Retrieving RecoveryManager from a ShardMapManager

The first step is to create a RecoveryManager instance. The [GetRecoveryManager method](#) returns the recovery manager for the current [ShardMapManager](#) instance. To address any inconsistencies in the shard map, you must first retrieve the RecoveryManager for the particular shard map.

```
ShardMapManager smm = ShardMapManagerFactory.GetSqlShardMapManager(smmConnectionString,
 ShardMapManagerLoadPolicy.Lazy);
RecoveryManager rm = smm.GetRecoveryManager();
```

In this example, the RecoveryManager is initialized from the ShardMapManager. The ShardMapManager containing a ShardMap is also already initialized.

Since this application code manipulates the shard map itself, the credentials used in the factory method (in the preceding example, smmConnectionString) should be credentials that have read-write permissions on the GSM database referenced by the connection string. These credentials are typically different from credentials used to open connections for data-dependent routing. For more information, see [Using credentials in the elastic database client](#).

## Removing a shard from the ShardMap after a shard is deleted

The [DetachShard method](#) detaches the given shard from the shard map and deletes mappings associated with the shard.

- The location parameter is the shard location, specifically server name and database name, of the shard being detached.
- The shardMapName parameter is the shard map name. This is only required when multiple shard maps are managed by the same shard map manager. Optional.

## IMPORTANT

Use this technique only if you are certain that the range for the updated mapping is empty. The methods above do not check data for the range being moved, so it is best to include checks in your code.

This example removes shards from the shard map.

```
rm.DetachShard(s.Location, customerMap);
```

The shard map reflects the shard location in the GSM before the deletion of the shard. Because the shard was deleted, it is assumed this was intentional, and the sharding key range is no longer in use. If not, you can execute point-in time restore. to recover the shard from an earlier point-in-time. (In that case, review the following section to detect shard inconsistencies.) To recover, see [Point in time recovery](#).

Since it is assumed the database deletion was intentional, the final administrative cleanup action is to delete the entry to the shard in the shard map manager. This prevents the application from inadvertently writing information to a range that is not expected.

## To detect mapping differences

The [DetectMappingDifferences method](#) selects and returns one of the shard maps (either local or global) as the source of truth and reconciles mappings on both shard maps (GSM and LSM).

```
rm.DetectMappingDifferences(location, shardMapName);
```

- The *location* specifies the server name and database name.
- The *shardMapName* parameter is the shard map name. This is only required if multiple shard maps are managed by the same shard map manager. Optional.

## To resolve mapping differences

The [ResolveMappingDifferences method](#) selects one of the shard maps (either local or global) as the source of truth and reconciles mappings on both shard maps (GSM and LSM).

```
ResolveMappingDifferences (RecoveryToken, MappingDifferenceResolution.KeepShardMapping);
```

- The *RecoveryToken* parameter enumerates the differences in the mappings between the GSM and the LSM for the specific shard.
- The [MappingDifferenceResolution enumeration](#) is used to indicate the method for resolving the difference between the shard mappings.
- **MappingDifferenceResolution.KeepShardMapping** is recommended that when the LSM contains the accurate mapping and therefore the mapping in the shard should be used. This is typically the case if there is a failover: the shard now resides on a new server. Since the shard must first be removed from the GSM (using the `RecoveryManager.DetachShard` method), a mapping no longer exists on the GSM. Therefore, the LSM must be used to re-establish the shard mapping.

## Attach a shard to the ShardMap after a shard is restored

The [AttachShard method](#) attaches the given shard to the shard map. It then detects any shard map inconsistencies and updates the mappings to match the shard at the point of the shard restoration. It is assumed that the database is also renamed to reflect the original database name (before the shard was restored), since the

point-in time restoration defaults to a new database appended with the timestamp.

```
rm.AttachShard(location, shardMapName)
```

- The *location* parameter is the server name and database name, of the shard being attached.
- The *shardMapName* parameter is the shard map name. This is only required when multiple shard maps are managed by the same shard map manager. Optional.

This example adds a shard to the shard map that has been recently restored from an earlier point-in time. Since the shard (namely the mapping for the shard in the LSM) has been restored, it is potentially inconsistent with the shard entry in the GSM. Outside of this example code, the shard was restored and renamed to the original name of the database. Since it was restored, it is assumed the mapping in the LSM is the trusted mapping.

```
rm.AttachShard(s.Location, customerMap);
var gs = rm.DetectMappingDifferences(s.Location);
foreach (RecoveryToken g in gs)
{
 rm.ResolveMappingDifferences(g, MappingDifferenceResolution.KeepShardMapping);
}
```

## Updating shard locations after a geo-failover (restore) of the shards

If there is a geo-failover, the secondary database is made write accessible and becomes the new primary database. The name of the server, and potentially the database (depending on your configuration), may be different from the original primary. Therefore the mapping entries for the shard in the GSM and LSM must be fixed. Similarly, if the database is restored to a different name or location, or to an earlier point in time, this might cause inconsistencies in the shard maps. The Shard Map Manager handles the distribution of open connections to the correct database. Distribution is based on the data in the shard map and the value of the sharding key that is the target of the applications request. After a geo-failover, this information must be updated with the accurate server name, database name and shard mapping of the recovered database.

## Best practices

Geo-failover and recovery are operations typically managed by a cloud administrator of the application intentionally utilizing one of Azure SQL databases business continuity features. Business continuity planning requires processes, procedures, and measures to ensure that business operations can continue without interruption. The methods available as part of the RecoveryManager class should be used within this work flow to ensure the GSM and LSM are kept up-to-date based on the recovery action taken. There are five basic steps to properly ensuring the GSM and LSM reflect the accurate information after a failover event. The application code to execute these steps can be integrated into existing tools and workflow.

1. Retrieve the RecoveryManager from the ShardMapManager.
2. Detach the old shard from the shard map.
3. Attach the new shard to the shard map, including the new shard location.
4. Detect inconsistencies in the mapping between the GSM and LSM.
5. Resolve differences between the GSM and the LSM, trusting the LSM.

This example performs the following steps:

1. Removes shards from the Shard Map that reflect shard locations before the failover event.
2. Attaches shards to the Shard Map reflecting the new shard locations (the parameter "Configuration.SecondaryServer" is the new server name but the same database name).

3. Retrieves the recovery tokens by detecting mapping differences between the GSM and the LSM for each shard.

4. Resolves the inconsistencies by trusting the mapping from the LSM of each shard.

```
var shards = smm.GetShards();
foreach (shard s in shards)
{
 if (s.Location.Server == Configuration.PrimaryServer)
 {
 ShardLocation slNew = new ShardLocation(Configuration.SecondaryServer, s.Location.Database);
 rm.DetachShard(s.Location);
 rm.AttachShard(slNew);
 var gs = rm.DetectMappingDifferences(slNew);
 foreach (RecoveryToken g in gs)
 {
 rm.ResolveMappingDifferences(g, MappingDifferenceResolution.KeepShardMapping);
 }
 }
}
```

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Migrate existing databases to scale out

11/22/2019 • 4 minutes to read • [Edit Online](#)

Easily manage your existing scaled-out sharded databases using Azure SQL Database database tools (such as the [Elastic Database client library](#)). First convert an existing set of databases to use the [shard map manager](#).

## Overview

To migrate an existing sharded database:

1. Prepare the [shard map manager database](#).
2. Create the shard map.
3. Prepare the individual shards.
4. Add mappings to the shard map.

These techniques can be implemented using either the [.NET Framework client library](#), or the PowerShell scripts found at [Azure SQL DB - Elastic Database tools scripts](#). The examples here use the PowerShell scripts.

For more information about the ShardMapManager, see [Shard map management](#). For an overview of the elastic database tools, see [Elastic Database features overview](#).

## Prepare the shard map manager database

The shard map manager is a special database that contains the data to manage scaled-out databases. You can use an existing database, or create a new database. A database acting as shard map manager should not be the same database as a shard. The PowerShell script does not create the database for you.

### Step 1: create a shard map manager

```
Create a shard map manager
New-ShardMapManager -UserName '<user_name>' -Password '<password>' -SqlServerName '<server_name>' -
SqlServerName '<smm_db_name>'

#<server_name> and <smm_db_name> are the server name and database name
for the new or existing database that should be used for storing
tenant-database mapping information.
```

#### To retrieve the shard map manager

After creation, you can retrieve the shard map manager with this cmdlet. This step is needed every time you need to use the ShardMapManager object.

```
Try to get a reference to the Shard Map Manager
$ShardMapManager = Get-ShardMapManager -UserName '<user_name>' -Password '<password>' -SqlServerName
'<server_name>' -SqlDatabaseName '<smm_db_name>'
```

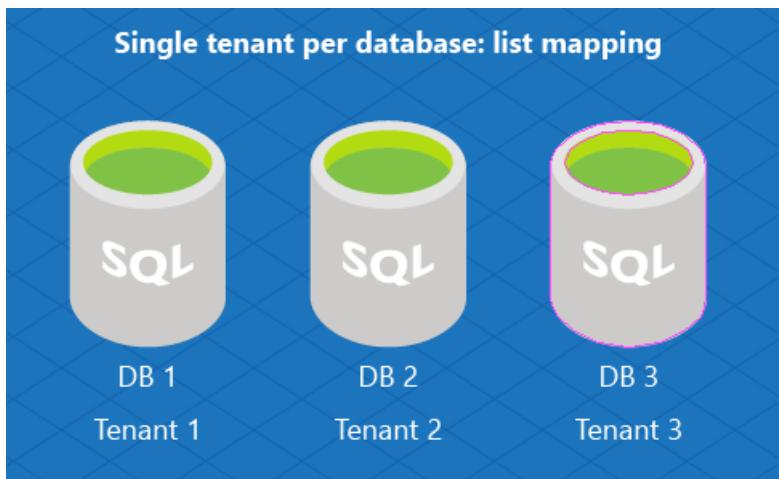
### Step 2: create the shard map

Select the type of shard map to create. The choice depends on the database architecture:

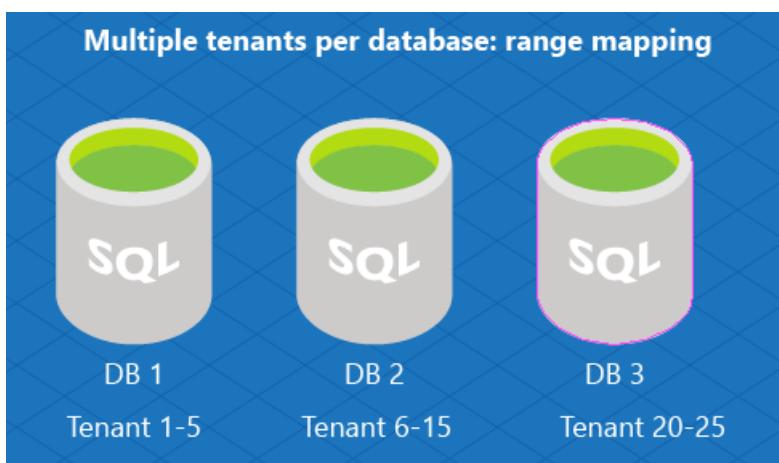
1. Single tenant per database (For terms, see the [glossary](#).)
2. Multiple tenants per database (two types):

- a. List mapping
- b. Range mapping

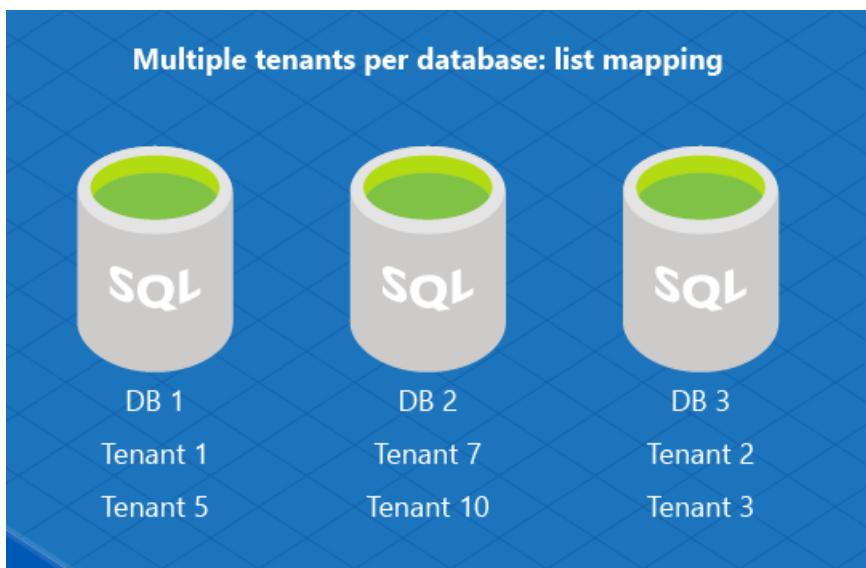
For a single-tenant model, create a **list mapping** shard map. The single-tenant model assigns one database per tenant. This is an effective model for SaaS developers as it simplifies management.



The multi-tenant model assigns several tenants to an individual database (and you can distribute groups of tenants across multiple databases). Use this model when you expect each tenant to have small data needs. In this model, assign a range of tenants to a database using **range mapping**.



Or you can implement a multi-tenant database model using a *list mapping* to assign multiple tenants to an individual database. For example, DB1 is used to store information about tenant ID 1 and 5, and DB2 stores data for tenant 7 and tenant 10.



**Based on your choice, choose one of these options:**

### **Option 1: create a shard map for a list mapping**

Create a shard map using the ShardMapManager object.

```
$ShardMapManager is the shard map manager object
$ShardMap = New-ListShardMap -KeyType $([int]) -ListShardMapName 'ListShardMap' -ShardMapManager
$ShardMapManager
```

### **Option 2: create a shard map for a range mapping**

To utilize this mapping pattern, tenant ID values needs to be continuous ranges, and it is acceptable to have gap in the ranges by skipping the range when creating the databases.

```
$ShardMapManager is the shard map manager object
'RangeShardMap' is the unique identifier for the range shard map.
$ShardMap = New-RangeShardMap -KeyType $([int]) -RangeShardMapName 'RangeShardMap' -ShardMapManager
$ShardMapManager
```

### **Option 3: List mappings on an individual database**

Setting up this pattern also requires creation of a list map as shown in step 2, option 1.

## **Step 3: Prepare individual shards**

Add each shard (database) to the shard map manager. This prepares the individual databases for storing mapping information. Execute this method on each shard.

```
Add-Shard -ShardMap $ShardMap -SqlServerName '<shard_server_name>' -SqlDatabaseName '<shard_database_name>'
The $ShardMap is the shard map created in step 2.
```

## **Step 4: Add mappings**

The addition of mappings depends on the kind of shard map you created. If you created a list map, you add list mappings. If you created a range map, you add range mappings.

### **Option 1: map the data for a list mapping**

Map the data by adding a list mapping for each tenant.

```
Create the mappings and associate it with the new shards
Add-ListMapping -KeyType $([int]) -ListPoint '<tenant_id>' -ListShardMap $ShardMap -SqlServerName
'<shard_server_name>' -SqlDatabaseName '<shard_database_name>'
```

### **Option 2: map the data for a range mapping**

Add the range mappings for all the tenant ID range - database associations:

```
Create the mappings and associate it with the new shards
Add-RangeMapping -KeyType $([int]) -RangeHigh '5' -RangeLow '1' -RangeShardMap $ShardMap -SqlServerName
'<shard_server_name>' -SqlDatabaseName '<shard_database_name>'
```

## **Step 4 option 3: map the data for multiple tenants on an individual database**

For each tenant, run the Add-ListMapping (option 1).

## Checking the mappings

Information about the existing shards and the mappings associated with them can be queried using following commands:

```
List the shards and mappings
Get-Shards -ShardMap $ShardMap
Get-Mappings -ShardMap $ShardMap
```

## Summary

Once you have completed the setup, you can begin to use the Elastic Database client library. You can also use [data-dependent routing](#) and [multi-shard query](#).

## Next steps

Get the PowerShell scripts from [Azure SQL DB-Elastic Database tools scripts](#).

The tools are also on GitHub: [Azure/elastic-db-tools](#).

Use the split-merge tool to move data to or from a multi-tenant model to a single tenant model. See [Split merge tool](#).

## Additional resources

For information on common data architecture patterns of multi-tenant software-as-a-service (SaaS) database applications, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).

## Questions and Feature Requests

For questions, use the [SQL Database forum](#) and for feature requests, add them to the [SQL Database feedback forum](#).

# Create performance counters to track performance of shard map manager

11/7/2019 • 2 minutes to read • [Edit Online](#)

Performance counters are used to track the performance of [data dependent routing](#) operations. These counters are accessible in the Performance Monitor, under the "Elastic Database: Shard Management" category.

You can capture the performance of a [shard map manager](#), especially when using [data dependent routing](#). Counters are created with methods of the `Microsoft.Azure.SqlDatabase.ElasticScale.Client` class.

**For the latest version:** Go to [Microsoft.Azure.SqlDatabase.ElasticScale.Client](#). See also [Upgrade an app to use the latest elastic database client library](#).

## Prerequisites

- To create the performance category and counters, the user must be a part of the local **Administrators** group on the machine hosting the application.
- To create a performance counter instance and update the counters, the user must be a member of either the **Administrators** or **Performance Monitor Users** group.

## Create performance category and counters

To create the counters, call the `CreatePerformanceCategoryAndCounters` method of the [ShardMapManagementFactory](#) class. Only an administrator can execute the method:

```
ShardMapManagerFactory.CreatePerformanceCategoryAndCounters()
```

You can also use [this](#) PowerShell script to execute the method. The method creates the following performance counters:

- **Cached mappings:** Number of mappings cached for the shard map.
- **DDR operations/sec:** Rate of data dependent routing operations for the shard map. This counter is updated when a call to `OpenConnectionForKey()` results in a successful connection to the destination shard.
- **Mapping lookup cache hits/sec:** Rate of successful cache lookup operations for mappings in the shard map.
- **Mapping lookup cache misses/sec:** Rate of failed cache lookup operations for mappings in the shard map.
- **Mappings added or updated in cache/sec:** Rate at which mappings are being added or updated in cache for the shard map.
- **Mappings removed from cache/sec:** Rate at which mappings are being removed from cache for the shard map.

Performance counters are created for each cached shard map per process.

## Notes

The following events trigger the creation of the performance counters:

- Initialization of the [ShardMapManager](#) with [eager loading](#), if the `ShardMapManager` contains any shard maps. These include the `GetSqlShardMapManager` and the `TryGetSqlShardMapManager` methods.
- Successful lookup of a shard map (using `GetShardMap()`, `GetListShardMap()` or `GetRangeShardMap()`).

- Successful creation of shard map using CreateShardMap().

The performance counters will be updated by all cache operations performed on the shard map and mappings. Successful removal of the shard map using DeleteShardMap() results in deletion of the performance counters instance.

## Best practices

- Creation of the performance category and counters should be performed only once before the creation of ShardMapManager object. Every execution of the command CreatePerformanceCategoryAndCounters() clears the previous counters (losing data reported by all instances) and creates new ones.
- Performance counter instances are created per process. Any application crash or removal of a shard map from the cache will result in deletion of the performance counters instances.

### See also

[Elastic Database features overview](#)

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Elastic Database client library with Entity Framework

2/7/2020 • 16 minutes to read • [Edit Online](#)

This document shows the changes in an Entity Framework application that are needed to integrate with the [Elastic Database tools](#). The focus is on composing [shard map management](#) and [data-dependent routing](#) with the Entity Framework **Code First** approach. The [Code First - New Database](#) tutorial for EF serves as the running example throughout this document. The sample code accompanying this document is part of elastic database tools' set of samples in the Visual Studio Code Samples.

## Downloading and Running the Sample Code

To download the code for this article:

- Visual Studio 2012 or later is required.
- Download the [Elastic DB Tools for Azure SQL - Entity Framework Integration sample](#). Unzip the sample to a location of your choosing.
- Start Visual Studio.
- In Visual Studio, select File -> Open Project/Solution.
- In the **Open Project** dialog, navigate to the sample you downloaded and select **EntityFrameworkCodeFirst.sln** to open the sample.

To run the sample, you need to create three empty databases in Azure SQL Database:

- Shard Map Manager database
- Shard 1 database
- Shard 2 database

Once you have created these databases, fill in the place holders in **Program.cs** with your Azure SQL DB server name, the database names, and your credentials to connect to the databases. Build the solution in Visual Studio. Visual Studio downloads the required NuGet packages for the elastic database client library, Entity Framework, and Transient Fault handling as part of the build process. Make sure that restoring NuGet packages is enabled for your solution. You can enable this setting by right-clicking on the solution file in the Visual Studio Solution Explorer.

## Entity Framework workflows

Entity Framework developers rely on one of the following four workflows to build applications and to ensure persistence for application objects:

- **Code First (New Database)**: The EF developer creates the model in the application code and then EF generates the database from it.
- **Code First (Existing Database)**: The developer lets EF generate the application code for the model from an existing database.
- **Model First**: The developer creates the model in the EF designer and then EF creates the database from the model.
- **Database First**: The developer uses EF tooling to infer the model from an existing database.

All these approaches rely on the `DbContext` class to transparently manage database connections and database schema for an application. Different constructors on the `DbContext` base class allow for different levels of control over connection creation, database bootstrapping, and schema creation. Challenges arise primarily from the fact

that the database connection management provided by EF intersects with the connection management capabilities of the data-dependent routing interfaces provided by the elastic database client library.

## Elastic database tools assumptions

For term definitions, see [Elastic Database tools glossary](#).

With elastic database client library, you define partitions of your application data called shardlets. Shardlets are identified by a sharding key and are mapped to specific databases. An application may have as many databases as needed and distribute the shardlets to provide enough capacity or performance given current business requirements. The mapping of sharding key values to the databases is stored by a shard map provided by the elastic database client APIs. This capability is called **Shard Map Management**, or SMM for short. The shard map also serves as the broker of database connections for requests that carry a sharding key. This capability is known as **data-dependent routing**.

The shard map manager protects users from inconsistent views into shardlet data that can occur when concurrent shardlet management operations (such as relocating data from one shard to another) are happening. To do so, the shard maps managed by the client library broker the database connections for an application. This allows the shard map functionality to automatically kill a database connection when shard management operations could impact the shardlet that the connection has been created for. This approach needs to integrate with some of EF's functionality, such as creating new connections from an existing one to check for database existence. In general, our observation has been that the standard DbContext constructors only work reliably for closed database connections that can safely be cloned for EF work. The design principle of elastic database instead is to only broker opened connections. One might think that closing a connection brokered by the client library before handing it over to the EF DbContext may solve this issue. However, by closing the connection and relying on EF to reopen it, one foregoes the validation and consistency checks performed by the library. The migrations functionality in EF, however, uses these connections to manage the underlying database schema in a way that is transparent to the application. Ideally, you will retain and combine all these capabilities from both the elastic database client library and EF in the same application. The following section discusses these properties and requirements in more detail.

## Requirements

When working with both the elastic database client library and Entity Framework APIs, you want to retain the following properties:

- **Scale-out:** To add or remove databases from the data tier of the sharded application as necessary for the capacity demands of the application. This means control over the creation and deletion of databases and using the elastic database shard map manager APIs to manage databases, and mappings of shardlets.
- **Consistency:** The application employs sharding, and uses the data-dependent routing capabilities of the client library. To avoid corruption or wrong query results, connections are brokered through the shard map manager. This also retains validation and consistency.
- **Code First:** To retain the convenience of EF's code first paradigm. In Code First, classes in the application are mapped transparently to the underlying database structures. The application code interacts with DbSets that mask most aspects involved in the underlying database processing.
- **Schema:** Entity Framework handles initial database schema creation and subsequent schema evolution through migrations. By retaining these capabilities, adapting your app is easy as the data evolves.

The following guidance instructs how to satisfy these requirements for Code First applications using elastic database tools.

## Data-dependent routing using EF DbContext

Database connections with Entity Framework are typically managed through subclasses of **DbContext**. Create

these subclasses by deriving from **DbContext**. This is where you define your **DbSets** that implement the database-backed collections of CLR objects for your application. In the context of data-dependent routing, you can identify several helpful properties that do not necessarily hold for other EF code first application scenarios:

- The database already exists and has been registered in the elastic database shard map.
- The schema of the application has already been deployed to the database (explained below).
- Data-dependent routing connections to the database are brokered by the shard map.

To integrate **DbContexts** with data-dependent routing for scale-out:

1. Create physical database connections through the elastic database client interfaces of the shard map manager,
2. Wrap the connection with the **DbContext** subclass
3. Pass the connection down into the **DbContext** base classes to ensure all the processing on the EF side happens as well.

The following code example illustrates this approach. (This code is also in the accompanying Visual Studio project)

```
public class ElasticScaleContext<T> : DbContext
{
 public DbSet<Blog> Blogs { get; set; }
 ...

 // C'tor for data-dependent routing. This call opens a validated connection
 // routed to the proper shard by the shard map manager.
 // Note that the base class c'tor call fails for an open connection
 // if migrations need to be done and SQL credentials are used. This is the reason for the
 // separation of c'tors into the data-dependent routing case (this c'tor) and the internal c'tor for new
 // shards.
 public ElasticScaleContext(ShardMap shardMap, T shardingKey, string connectionString)
 : base(CreateDDRConnection(shardMap, shardingKey, connectionString),
 true /* contextOwnsConnection */)
 {
 }

 // Only static methods are allowed in calls into base class c'tors.
 private static DbConnection CreateDDRConnection(
 ShardMap shardMap,
 T shardingKey,
 string connectionString)
 {
 // No initialization
 Database.SetInitializer<ElasticScaleContext<T>>(null);

 // Ask shard map to broker a validated connection for the given key
 SqlConnection conn = shardMap.OpenConnectionForKey<T>
 (shardingKey, connectionString, ConnectionOptions.Validate);
 return conn;
 }
}
```

## Main points

- A new constructor replaces the default constructor in the **DbContext** subclass
- The new constructor takes the arguments that are required for data-dependent routing through elastic database client library:
  - the shard map to access the data-dependent routing interfaces,
  - the sharding key to identify the shardlet,
  - a connection string with the credentials for the data-dependent routing connection to the shard.
- The call to the base class constructor takes a detour into a static method that performs all the steps

necessary for data-dependent routing.

- It uses the OpenConnectionForKey call of the elastic database client interfaces on the shard map to establish an open connection.
- The shard map creates the open connection to the shard that holds the shardlet for the given sharding key.
- This open connection is passed back to the base class constructor of DbContext to indicate that this connection is to be used by EF instead of letting EF create a new connection automatically. This way the connection has been tagged by the elastic database client API so that it can guarantee consistency under shard map management operations.

Use the new constructor for your DbContext subclass instead of the default constructor in your code. Here is an example:

```
// Create and save a new blog.

Console.WriteLine("Enter a name for a new blog: ");
var name = Console.ReadLine();

using (var db = new ElasticScaleContext<int>(
 sharding.ShardMap,
 tenantId1,
 connStrBldr.ConnectionString))
{
 var blog = new Blog { Name = name };
 db.Blogs.Add(blog);
 db.SaveChanges();

 // Display all Blogs for tenant 1
 var query = from b in db.Blogs
 orderby b.Name
 select b;
 ...
}
```

The new constructor opens the connection to the shard that holds the data for the shardlet identified by the value of **tenantid1**. The code in the **using** block stays unchanged to access the **DbSet** for blogs using EF on the shard for **tenantid1**. This changes semantics for the code in the using block such that all database operations are now scoped to the one shard where **tenantid1** is kept. For instance, a LINQ query over the blogs **DbSet** would only return blogs stored on the current shard, but not the ones stored on other shards.

#### Transient faults handling

The Microsoft Patterns & Practices team published the [The Transient Fault Handling Application Block](#). The library is used with elastic scale client library in combination with EF. However, ensure that any transient exception returns to a place where you can ensure that the new constructor is being used after a transient fault so that any new connection attempt is made using the constructors you tweaked. Otherwise, a connection to the correct shard is not guaranteed, and there are no assurances the connection is maintained as changes to the shard map occur.

The following code sample illustrates how a SQL retry policy can be used around the new **DbContext** subclass constructors:

```

SqlDatabaseUtils.SqlRetryPolicy.ExecuteAction(() =>
{
 using (var db = new ElasticScaleContext<int>(
 sharding.ShardMap,
 tenantId1,
 connStrBldr.ConnectionString))
 {
 var blog = new Blog { Name = name };
 db.Blogs.Add(blog);
 db.SaveChanges();
 ...
 }
});

```

**SqlDatabaseUtils.SqlRetryPolicy** in the code above is defined as a **SqlDatabaseTransientErrorDetectionStrategy** with a retry count of 10, and 5 seconds wait time between retries. This approach is similar to the guidance for EF and user-initiated transactions (see [Limitations with Retrying Execution Strategies \(EF6 onwards\)](#)). Both situations require that the application program controls the scope to which the transient exception returns: to either reopen the transaction, or (as shown) recreate the context from the proper constructor that uses the elastic database client library.

The need to control where transient exceptions take us back in scope also precludes the use of the built-in **SqlAzureExecutionStrategy** that comes with EF. **SqlAzureExecutionStrategy** would reopen a connection but not use **OpenConnectionForKey** and therefore bypass all the validation that is performed as part of the **OpenConnectionForKey** call. Instead, the code sample uses the built-in **DefaultExecutionStrategy** that also comes with EF. As opposed to **SqlAzureExecutionStrategy**, it works correctly in combination with the retry policy from Transient Fault Handling. The execution policy is set in the **ElasticScaleDbConfiguration** class. Note that we decided not to use **DefaultSqlExecutionStrategy** since it suggests using **SqlAzureExecutionStrategy** if transient exceptions occur - which would lead to wrong behavior as discussed. For more information on the different retry policies and EF, see [Connection Resiliency in EF](#).

#### Constructor rewrites

The code examples above illustrate the default constructor re-writes required for your application in order to use data-dependent routing with the Entity Framework. The following table generalizes this approach to other constructors.

CURRENT CONSTRUCTOR	REWRITTEN CONSTRUCTOR FOR DATA	BASE CONSTRUCTOR	NOTES
MyContext()	ElasticScaleContext(ShardMap, TKey)	DbContext(DbConnection, bool)	The connection needs to be a function of the shard map and the data-dependent routing key. You need to bypass automatic connection creation by EF and instead use the shard map to broker the connection.
MyContext(string)	ElasticScaleContext(ShardMap, TKey)	DbContext(DbConnection, bool)	The connection is a function of the shard map and the data-dependent routing key. A fixed database name or connection string does not work as they by-pass validation by the shard map.

CURRENT CONSTRUCTOR	REWRITTEN CONSTRUCTOR FOR DATA	BASE CONSTRUCTOR	NOTES
MyContext(DbCompiledModel)	ElasticScaleContext(ShardMap, TKey, DbCompiledModel)	DbContext(DbConnection, DbCompiledModel, bool)	The connection gets created for the given shard map and sharding key with the model provided. The compiled model is passed on to the base c'tor.
MyContext(DbConnection, bool)	ElasticScaleContext(ShardMap, TKey, bool)	DbContext(DbConnection, bool)	The connection needs to be inferred from the shard map and the key. It cannot be provided as an input (unless that input was already using the shard map and the key). The Boolean is passed on.
MyContext(string, DbCompiledModel)	ElasticScaleContext(ShardMap, TKey, DbCompiledModel)	DbContext(DbConnection, DbCompiledModel, bool)	The connection needs to be inferred from the shard map and the key. It cannot be provided as an input (unless that input was using the shard map and the key). The compiled model is passed on.
MyContext(ObjectContext, bool)	ElasticScaleContext(ShardMap, TKey, ObjectContext, bool)	DbContext(ObjectContext, bool)	The new constructor needs to ensure that any connection in the ObjectContext passed as an input is re-routed to a connection managed by Elastic Scale. A detailed discussion of ObjectContexts is beyond the scope of this document.
MyContext(DbConnection, DbCompiledModel, bool)	ElasticScaleContext(ShardMap, TKey, DbCompiledModel, bool)	DbContext(DbConnection, DbCompiledModel, bool);	The connection needs to be inferred from the shard map and the key. The connection cannot be provided as an input (unless that input was already using the shard map and the key). Model and Boolean are passed on to the base class constructor.

## Shard schema deployment through EF migrations

Automatic schema management is a convenience provided by the Entity Framework. In the context of applications using elastic database tools, you want to retain this capability to automatically provision the schema to newly created shards when databases are added to the sharded application. The primary use case is to increase capacity at the data tier for sharded applications using EF. Relying on EF's capabilities for schema management reduces the database administration effort with a sharded application built on EF.

Schema deployment through EF migrations works best on **unopened connections**. This is in contrast to the scenario for data-dependent routing that relies on the opened connection provided by the elastic database client API. Another difference is the consistency requirement: While desirable to ensure consistency for all data-

dependent routing connections to protect against concurrent shard map manipulation, it is not a concern with initial schema deployment to a new database that has not yet been registered in the shard map, and not yet been allocated to hold shardlets. You can therefore rely on regular database connections for this scenario, as opposed to data-dependent routing.

This leads to an approach where schema deployment through EF migrations is tightly coupled with the registration of the new database as a shard in the application's shard map. This relies on the following prerequisites:

- The database has already been created.
- The database is empty - it holds no user schema and no user data.
- The database cannot yet be accessed through the elastic database client APIs for data-dependent routing.

With these prerequisites in place, you can create a regular un-opened **SqlConnection** to kick off EF migrations for schema deployment. The following code sample illustrates this approach.

```
// Enter a new shard - i.e. an empty database - to the shard map, allocate a first tenant to it
// and kick off EF initialization of the database to deploy schema

public void RegisterNewShard(string server, string database, string connStr, int key)
{

 Shard shard = this.ShardMap.CreateShard(new ShardLocation(server, database));

 SqlConnectionStringBuilder connStrBldr = new SqlConnectionStringBuilder(connStr);
 connStrBldr.DataSource = server;
 connStrBldr.InitialCatalog = database;

 // Go into a DbContext to trigger migrations and schema deployment for the new shard.
 // This requires an un-opened connection.
 using (var db = new ElasticScaleContext<int>(connStrBldr.ConnectionString))
 {
 // Run a query to engage EF migrations
 (from b in db.Blogs
 select b).Count();
 }

 // Register the mapping of the tenant to the shard in the shard map.
 // After this step, data-dependent routing on the shard map can be used

 this.ShardMap.CreatePointMapping(key, shard);
}
```

This sample shows the method **RegisterNewShard** that registers the shard in the shard map, deploys the schema through EF migrations, and stores a mapping of a sharding key to the shard. It relies on a constructor of the **DbContext** subclass (**ElasticScaleContext** in the sample) that takes a SQL connection string as input. The code of this constructor is straight-forward, as the following example shows:

```

// C'tor to deploy schema and migrations to a new shard
protected internal ElasticScaleContext(string connectionString)
 : base(SetInitializerForConnection(connectionString))
{
}

// Only static methods are allowed in calls into base class c'tors
private static string SetInitializerForConnection(string connectionString)
{
 // You want existence checks so that the schema can get deployed
 Database.SetInitializer<ElasticScaleContext<T>>(
new CreateDatabaseIfNotExists<ElasticScaleContext<T>>());

 return connectionString;
}

```

One might have used the version of the constructor inherited from the base class. But the code needs to ensure that the default initializer for EF is used when connecting. Hence the short detour into the static method before calling into the base class constructor with the connection string. Note that the registration of shards should run in a different app domain or process to ensure that the initializer settings for EF do not conflict.

## Limitations

The approaches outlined in this document entail a couple of limitations:

- EF applications that use **LocalDb** first need to migrate to a regular SQL Server database before using elastic database client library. Scaling out an application through sharding with Elastic Scale is not possible with **LocalDb**. Note that development can still use **LocalDb**.
- Any changes to the application that imply database schema changes need to go through EF migrations on all shards. The sample code for this document does not demonstrate how to do this. Consider using Update-Database with a ConnectionString parameter to iterate over all shards; or extract the T-SQL script for the pending migration using Update-Database with the -Script option and apply the T-SQL script to your shards.
- Given a request, it is assumed that all of its database processing is contained within a single shard as identified by the sharding key provided by the request. However, this assumption does not always hold true. For example, when it is not possible to make a sharding key available. To address this, the client library provides the **MultiShardQuery** class that implements a connection abstraction for querying over several shards. Learning to use the **MultiShardQuery** in combination with EF is beyond the scope of this document

## Conclusion

Through the steps outlined in this document, EF applications can use the elastic database client library's capability for data-dependent routing by refactoring constructors of the **DbContext** subclasses used in the EF application. This limits the changes required to those places where **DbContext** classes already exist. In addition, EF applications can continue to benefit from automatic schema deployment by combining the steps that invoke the necessary EF migrations with the registration of new shards and mappings in the shard map.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Using elastic database client library with Dapper

11/7/2019 • 8 minutes to read • [Edit Online](#)

This document is for developers that rely on Dapper to build applications, but also want to embrace [elastic database tooling](#) to create applications that implement sharding to scale out their data tier. This document illustrates the changes in Dapper-based applications that are necessary to integrate with elastic database tools. Our focus is on composing the elastic database shard management and data-dependent routing with Dapper.

**Sample Code:** [Elastic database tools for Azure SQL Database - Dapper integration](#).

Integrating **Dapper** and **DapperExtensions** with the elastic database client library for Azure SQL Database is easy. Your applications can use data-dependent routing by changing the creation and opening of new `SqlConnection` objects to use the `OpenConnectionForKey` call from the [client library](#). This limits changes in your application to only where new connections are created and opened.

## Dapper overview

**Dapper** is an object-relational mapper. It maps .NET objects from your application to a relational database (and vice versa). The first part of the sample code illustrates how you can integrate the elastic database client library with Dapper-based applications. The second part of the sample code illustrates how to integrate when using both Dapper and DapperExtensions.

The mapper functionality in Dapper provides extension methods on database connections that simplify submitting T-SQL statements for execution or querying the database. For instance, Dapper makes it easy to map between your .NET objects and the parameters of SQL statements for **Execute** calls, or to consume the results of your SQL queries into .NET objects using **Query** calls from Dapper.

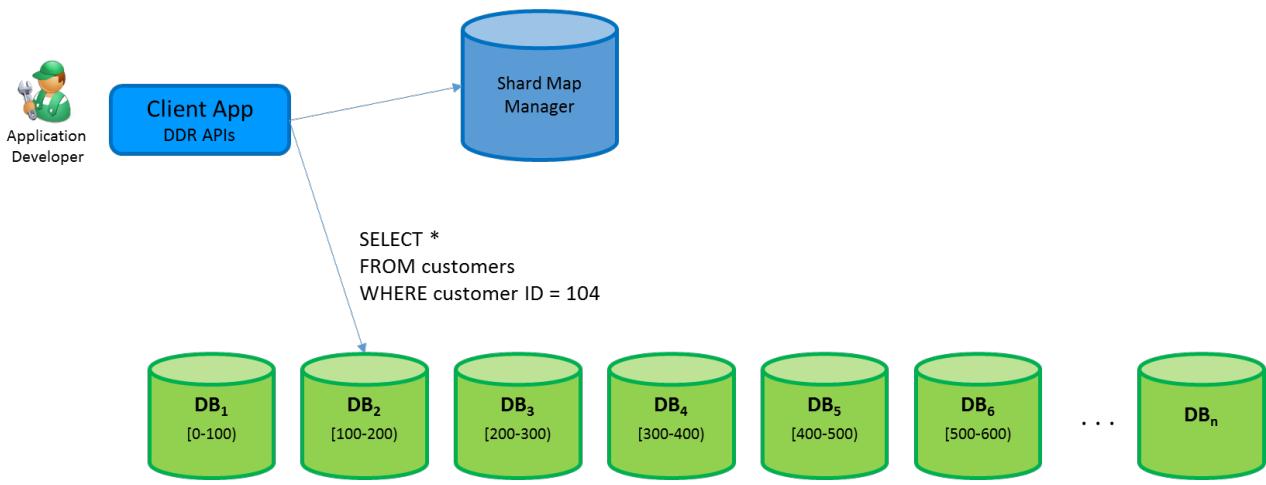
When using DapperExtensions, you no longer need to provide the SQL statements. Extensions methods such as **GetList** or **Insert** over the database connection create the SQL statements behind the scenes.

Another benefit of Dapper and also DapperExtensions is that the application controls the creation of the database connection. This helps interact with the elastic database client library which brokers database connections based on the mapping of shardlets to databases.

To get the Dapper assemblies, see [Dapper dot net](#). For the Dapper extensions, see [DapperExtensions](#).

## A quick Look at the elastic database client library

With the elastic database client library, you define partitions of your application data called *shardlets*, map them to databases, and identify them by *sharding keys*. You can have as many databases as you need and distribute your shardlets across these databases. The mapping of sharding key values to the databases is stored by a shard map provided by the library's APIs. This capability is called **shard map management**. The shard map also serves as the broker of database connections for requests that carry a sharding key. This capability is referred to as **data-dependent routing**.



The shard map manager protects users from inconsistent views into shardlet data that can occur when concurrent shardlet management operations are happening on the databases. To do so, the shard maps broker the database connections for an application built with the library. When shard management operations could impact the shardlet, this allows the shard map functionality to automatically kill a database connection.

Instead of using the traditional way to create connections for Dapper, you need to use the [OpenConnectionForKey method](#). This ensures that all the validation takes place and connections are managed properly when any data moves between shards.

### Requirements for Dapper integration

When working with both the elastic database client library and the Dapper APIs, you want to retain the following properties:

- **Scale out:** We want to add or remove databases from the data tier of the sharded application as necessary for the capacity demands of the application.
- **Consistency:** Since the application is scaled out using sharding, you need to perform data-dependent routing. We want to use the data-dependent routing capabilities of the library to do so. In particular, you want to retain the validation and consistency guarantees provided by connections that are brokered through the shard map manager in order to avoid corruption or wrong query results. This ensures that connections to a given shardlet are rejected or stopped if (for instance) the shardlet is currently moved to a different shard using Split/Merge APIs.
- **Object Mapping:** We want to retain the convenience of the mappings provided by Dapper to translate between classes in the application and the underlying database structures.

The following section provides guidance for these requirements for applications based on [Dapper](#) and [DapperExtensions](#).

## Technical Guidance

### Data-dependent routing with Dapper

With Dapper, the application is typically responsible for creating and opening the connections to the underlying database. Given a type T by the application, Dapper returns query results as .NET collections of type T. Dapper performs the mapping from the T-SQL result rows to the objects of type T. Similarly, Dapper maps .NET objects into SQL values or parameters for data manipulation language (DML) statements. Dapper offers this functionality via extension methods on the regular [SqlConnection](#) object from the ADO .NET SQL Client libraries. The SQL connection returned by the Elastic Scale APIs for DDR are also regular [SqlConnection](#) objects. This allows us to directly use Dapper extensions over the type returned by the client library's DDR API, as it is also a simple SQL Client connection.

These observations make it straightforward to use connections brokered by the elastic database client library for Dapper.

This code example (from the accompanying sample) illustrates the approach where the sharding key is provided by the application to broker the connection to the right shard.

```
using (SqlConnection sqlconn = shardingLayer.ShardMap.OpenConnectionForKey(
 key: tenantId1,
 connectionString: connStrBldr.ConnectionString,
 options: ConnectionOptions.Validate))
{
 var blog = new Blog { Name = name };
 sqlconn.Execute(@"
 INSERT INTO
 Blog (Name)
 VALUES (@name)", new { name = blog.Name })
};
```

The call to the [OpenConnectionForKey](#) API replaces the default creation and opening of a SQL Client connection. The [OpenConnectionForKey](#) call takes the arguments that are required for data-dependent routing:

- The shard map to access the data-dependent routing interfaces
- The sharding key to identify the shardlet
- The credentials (user name and password) to connect to the shard

The shard map object creates a connection to the shard that holds the shardlet for the given sharding key. The elastic database client APIs also tag the connection to implement its consistency guarantees. Since the call to [OpenConnectionForKey](#) returns a regular SQL Client connection object, the subsequent call to the **Execute** extension method from Dapper follows the standard Dapper practice.

Queries work very much the same way - you first open the connection using [OpenConnectionForKey](#) from the client API. Then you use the regular Dapper extension methods to map the results of your SQL query into .NET objects:

```
using (SqlConnection sqlconn = shardingLayer.ShardMap.OpenConnectionForKey(
 key: tenantId1,
 connectionString: connStrBldr.ConnectionString,
 options: ConnectionOptions.Validate))
{
 // Display all Blogs for tenant 1
 IEnumerable<Blog> result = sqlconn.Query<Blog>(@"
 SELECT *
 FROM Blog
 ORDER BY Name");

 Console.WriteLine("All blogs for tenant id {0}:", tenantId1);
 foreach (var item in result)
 {
 Console.WriteLine(item.Name);
 }
}
```

Note that the **using** block with the DDR connection scopes all database operations within the block to the one shard where tenantId1 is kept. The query only returns blogs stored on the current shard, but not the ones stored on any other shards.

## Data-dependent routing with Dapper and DapperExtensions

Dapper comes with an ecosystem of additional extensions that can provide further convenience and abstraction from the database when developing database applications. DapperExtensions is an example.

Using DapperExtensions in your application does not change how database connections are created and managed. It is still the application's responsibility to open connections, and regular SQL Client connection objects are expected by the extension methods. We can rely on the [OpenConnectionForKey](#) as outlined above. As the following code samples show, the only change is that you no longer have to write the T-SQL statements:

```
using (SqlConnection sqlconn = shardingLayer.ShardMap.OpenConnectionForKey(
 key: tenantId2,
 connectionString: connStrBldr.ConnectionString,
 options: ConnectionOptions.Validate))
{
 var blog = new Blog { Name = name2 };
 sqlconn.Insert(blog);
}
```

And here is the code sample for the query:

```
using (SqlConnection sqlconn = shardingLayer.ShardMap.OpenConnectionForKey(
 key: tenantId2,
 connectionString: connStrBldr.ConnectionString,
 options: ConnectionOptions.Validate))
{
 // Display all Blogs for tenant 2
 IEnumerable<Blog> result = sqlconn.GetList<Blog>();
 Console.WriteLine("All blogs for tenant id {0}:", tenantId2);
 foreach (var item in result)
 {
 Console.WriteLine(item.Name);
 }
}
```

## Handling transient faults

The Microsoft Patterns & Practices team published the [Transient Fault Handling Application Block](#) to help application developers mitigate common transient fault conditions encountered when running in the cloud. For more information, see [Perseverance, Secret of All Triumphs: Using the Transient Fault Handling Application Block](#).

The code sample relies on the transient fault library to protect against transient faults.

```
SqlDatabaseUtils.SqlRetryPolicy.ExecuteAction(() =>
{
 using (SqlConnection sqlconn =
 shardingLayer.ShardMap.OpenConnectionForKey(tenantId2, connStrBldr.ConnectionString,
 ConnectionOptions.Validate))
 {
 var blog = new Blog { Name = name2 };
 sqlconn.Insert(blog);
 }
});
```

**SqlDatabaseUtils.SqlRetryPolicy** in the code above is defined as a **SqlDatabaseTransientErrorDetectionStrategy** with a retry count of 10, and 5 seconds wait time between retries. If you are using transactions, make sure that your retry scope goes back to the beginning of the transaction in the case of a transient fault.

## Limitations

The approaches outlined in this document entail a couple of limitations:

- The sample code for this document does not demonstrate how to manage schema across shards.

- Given a request, we assume that all its database processing is contained within a single shard as identified by the sharding key provided by the request. However, this assumption does not always hold, for example, when it is not possible to make a sharding key available. To address this, the elastic database client library includes the [MultiShardQuery class](#). The class implements a connection abstraction for querying over several shards. Using MultiShardQuery in combination with Dapper is beyond the scope of this document.

## Conclusion

Applications using Dapper and DapperExtensions can easily benefit from elastic database tools for Azure SQL Database. Through the steps outlined in this document, those applications can use the tool's capability for data-dependent routing by changing the creation and opening of new [SqlConnection](#) objects to use the [OpenConnectionForKey](#) call of the elastic database client library. This limits the application changes required to those places where new connections are created and opened.

## Additional resources

Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Get started with cross-database queries (vertical partitioning) (preview)

11/7/2019 • 2 minutes to read • [Edit Online](#)

Elastic database query (preview) for Azure SQL Database allows you to run T-SQL queries that span multiple databases using a single connection point. This article applies to [vertically partitioned databases](#).

When completed, you will learn how to configure and use an Azure SQL Database to perform queries that span multiple related databases.

For more information about the elastic database query feature, see [Azure SQL Database elastic database query overview](#).

## Prerequisites

ALTER ANY EXTERNAL DATA SOURCE permission is required. This permission is included with the ALTER DATABASE permission. ALTER ANY EXTERNAL DATA SOURCE permissions are needed to refer to the underlying data source.

## Create the sample databases

To start with, create two databases, **Customers** and **Orders**, either in the same or different SQL Database servers.

Execute the following queries on the **Orders** database to create the **OrderInformation** table and input the sample data.

```
CREATE TABLE [dbo].[OrderInformation](
 [OrderID] [int] NOT NULL,
 [CustomerID] [int] NOT NULL
)
INSERT INTO [dbo].[OrderInformation] ([OrderID], [CustomerID]) VALUES (123, 1)
INSERT INTO [dbo].[OrderInformation] ([OrderID], [CustomerID]) VALUES (149, 2)
INSERT INTO [dbo].[OrderInformation] ([OrderID], [CustomerID]) VALUES (857, 2)
INSERT INTO [dbo].[OrderInformation] ([OrderID], [CustomerID]) VALUES (321, 1)
INSERT INTO [dbo].[OrderInformation] ([OrderID], [CustomerID]) VALUES (564, 8)
```

Now, execute following query on the **Customers** database to create the **CustomerInformation** table and input the sample data.

```
CREATE TABLE [dbo].[CustomerInformation](
 [CustomerID] [int] NOT NULL,
 [CustomerName] [varchar](50) NULL,
 [Company] [varchar](50) NULL
 CONSTRAINT [CustID] PRIMARY KEY CLUSTERED ([CustomerID] ASC)
)
INSERT INTO [dbo].[CustomerInformation] ([CustomerID], [CustomerName], [Company]) VALUES (1, 'Jack', 'ABC')
INSERT INTO [dbo].[CustomerInformation] ([CustomerID], [CustomerName], [Company]) VALUES (2, 'Steve', 'XYZ')
INSERT INTO [dbo].[CustomerInformation] ([CustomerID], [CustomerName], [Company]) VALUES (3, 'Lylla', 'MNO')
```

## Create database objects

### Database scoped master key and credentials

1. Open SQL Server Management Studio or SQL Server Data Tools in Visual Studio.
2. Connect to the Orders database and execute the following T-SQL commands:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<master_key_password>';
CREATE DATABASE SCOPED CREDENTIAL ElasticDBQueryCred
WITH IDENTITY = '<username>',
SECRET = '<password>';
```

The "username" and "password" should be the username and password used to log in into the Customers database. Authentication using Azure Active Directory with elastic queries is not currently supported.

## External data sources

To create an external data source, execute the following command on the Orders database:

```
CREATE EXTERNAL DATA SOURCE MyElasticDBQueryDataSrc WITH
 (TYPE = RDBMS,
 LOCATION = '<server_name>.database.windows.net',
 DATABASE_NAME = 'Customers',
 CREDENTIAL = ElasticDBQueryCred,
) ;
```

## External tables

Create an external table on the Orders database, which matches the definition of the CustomerInformation table:

```
CREATE EXTERNAL TABLE [dbo].[CustomerInformation]
([CustomerID] [int] NOT NULL,
 [CustomerName] [varchar](50) NOT NULL,
 [Company] [varchar](50) NOT NULL)
WITH
(DATA_SOURCE = MyElasticDBQueryDataSrc)
```

## Execute a sample elastic database T-SQL query

Once you have defined your external data source and your external tables, you can now use T-SQL to query your external tables. Execute this query on the Orders database:

```
SELECT OrderInformation.CustomerID, OrderInformation.OrderId, CustomerInformation.CustomerName,
CustomerInformation.Company
FROM OrderInformation
INNER JOIN CustomerInformation
ON CustomerInformation.CustomerID = OrderInformation.CustomerID
```

## Cost

Currently, the elastic database query feature is included into the cost of your Azure SQL Database.

For pricing information, see [SQL Database Pricing](#).

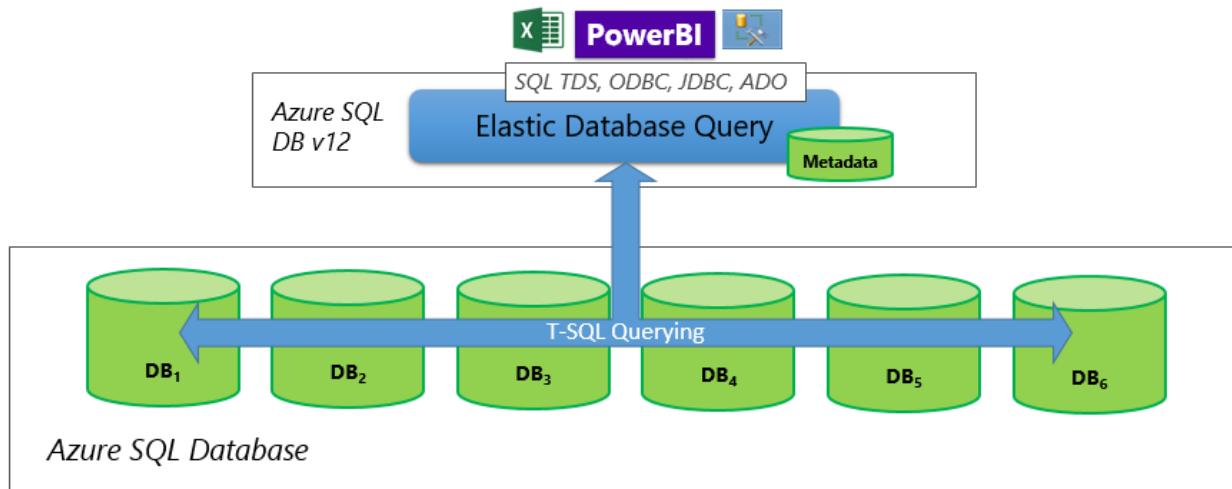
## Next steps

- For an overview of elastic query, see [Elastic query overview](#).
- For syntax and sample queries for vertically partitioned data, see [Querying vertically partitioned data](#))
- For a horizontal partitioning (sharding) tutorial, see [Getting started with elastic query for horizontal partitioning \(sharding\)](#).

- For syntax and sample queries for horizontally partitioned data, see [Querying horizontally partitioned data](#))
- See [sp\\_execute\\_remote](#) for a stored procedure that executes a Transact-SQL statement on a single remote Azure SQL Database or set of databases serving as shards in a horizontal partitioning scheme.

# Reporting across scaled-out cloud databases (preview)

11/7/2019 • 7 minutes to read • [Edit Online](#)



Sharded databases distribute rows across a scaled out data tier. The schema is identical on all participating databases, also known as horizontal partitioning. Using an elastic query, you can create reports that span all databases in a sharded database.

For a quick start, see [Reporting across scaled-out cloud databases](#).

For non-sharded databases, see [Query across cloud databases with different schemas](#).

## Prerequisites

- Create a shard map using the elastic database client library. see [Shard map management](#). Or use the sample app in [Get started with elastic database tools](#).
- Alternatively, see [Migrate existing databases to scaled-out databases](#).
- The user must possess ALTER ANY EXTERNAL DATA SOURCE permission. This permission is included with the ALTER DATABASE permission.
- ALTER ANY EXTERNAL DATA SOURCE permissions are needed to refer to the underlying data source.

## Overview

These statements create the metadata representation of your sharded data tier in the elastic query database.

1. [CREATE MASTER KEY](#)
2. [CREATE DATABASE SCOPED CREDENTIAL](#)
3. [CREATE EXTERNAL DATA SOURCE](#)
4. [CREATE EXTERNAL TABLE](#)

### 1.1 Create database scoped master key and credentials

The credential is used by the elastic query to connect to your remote databases.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'password';
CREATE DATABASE SCOPED CREDENTIAL <credential_name> WITH IDENTITY = '<username>',
SECRET = '<password>'
[;]
```

#### NOTE

Make sure that the "<username>" does not include any "@servername" suffix.

## 1.2 Create external data sources

Syntax:

```
<External_Data_Source> ::=
CREATE EXTERNAL DATA SOURCE <data_source_name> WITH
 (TYPE = SHARD_MAP_MANAGER,
 LOCATION = '<fully_qualified_server_name>',
 DATABASE_NAME = '<shardmap_database_name>',
 CREDENTIAL = <credential_name>,
 SHARD_MAP_NAME = '<shardmapname>'
) [;]
```

#### Example

```
CREATE EXTERNAL DATA SOURCE MyExtSrc
WITH
(
 TYPE=SHARD_MAP_MANAGER,
 LOCATION='myserver.database.windows.net',
 DATABASE_NAME='ShardMapDatabase',
 CREDENTIAL= SMMUser,
 SHARD_MAP_NAME='ShardMap'
);
```

Retrieve the list of current external data sources:

```
select * from sys.external_data_sources;
```

The external data source references your shard map. An elastic query then uses the external data source and the underlying shard map to enumerate the databases that participate in the data tier. The same credentials are used to read the shard map and to access the data on the shards during the processing of an elastic query.

## 1.3 Create external tables

Syntax:

```

CREATE EXTERNAL TABLE [database_name . [schema_name] . | schema_name.] table_name
({ <column_definition> } [,...n]
{ WITH (<sharded_external_table_options>) }
) [;]

<sharded_external_table_options> ::==
 DATA_SOURCE = <External_Data_Source>,
 [SCHEMA_NAME = N'nonescaped_schema_name',]
 [OBJECT_NAME = N'nonescaped_object_name',]
 DISTRIBUTION = SHARDED(<sharding_column_name>) | REPLICATED |ROUND_ROBIN

```

## Example

```

CREATE EXTERNAL TABLE [dbo].[order_line](
 [ol_o_id] int NOT NULL,
 [ol_d_id] tinyint NOT NULL,
 [ol_w_id] int NOT NULL,
 [ol_number] tinyint NOT NULL,
 [ol_i_id] int NOT NULL,
 [ol_delivery_d] datetime NOT NULL,
 [ol_amount] smallmoney NOT NULL,
 [ol_supply_w_id] int NOT NULL,
 [ol_quantity] smallint NOT NULL,
 [ol_dist_info] char(24) NOT NULL
)

WITH
(
 DATA_SOURCE = MyExtSrc,
 SCHEMA_NAME = 'orders',
 OBJECT_NAME = 'order_details',
 DISTRIBUTION=SHARDED(ol_w_id)
);

```

Retrieve the list of external tables from the current database:

```
SELECT * from sys.external_tables;
```

To drop external tables:

```
DROP EXTERNAL TABLE [database_name . [schema_name] . | schema_name.] table_name[;]
```

## Remarks

The DATA\_SOURCE clause defines the external data source (a shard map) that is used for the external table.

The SCHEMA\_NAME and OBJECT\_NAME clauses map the external table definition to a table in a different schema. If omitted, the schema of the remote object is assumed to be "dbo" and its name is assumed to be identical to the external table name being defined. This is useful if the name of your remote table is already taken in the database where you want to create the external table. For example, you want to define an external table to get an aggregate view of catalog views or DMVs on your scaled out data tier. Since catalog views and DMVs already exist locally, you cannot use their names for the external table definition. Instead, use a different name and use the catalog view's or the DMV's name in the SCHEMA\_NAME and/or OBJECT\_NAME clauses. (See the example below.)

The DISTRIBUTION clause specifies the data distribution used for this table. The query processor utilizes the information provided in the DISTRIBUTION clause to build the most efficient query plans.

1. **SHARDED** means data is horizontally partitioned across the databases. The partitioning key for the data

distribution is the **<sharding\_column\_name>** parameter.

2. **REPLICATED** means that identical copies of the table are present on each database. It is your responsibility to ensure that the replicas are identical across the databases.
3. **ROUND\_ROBIN** means that the table is horizontally partitioned using an application-dependent distribution method.

**Data tier reference:** The external table DDL refers to an external data source. The external data source specifies a shard map which provides the external table with the information necessary to locate all the databases in your data tier.

### Security considerations

Users with access to the external table automatically gain access to the underlying remote tables under the credential given in the external data source definition. Avoid undesired elevation of privileges through the credential of the external data source. Use GRANT or REVOKE for an external table just as though it were a regular table.

Once you have defined your external data source and your external tables, you can now use full T-SQL over your external tables.

## Example: querying horizontal partitioned databases

The following query performs a three-way join between warehouses, orders and order lines and uses several aggregates and a selective filter. It assumes (1) horizontal partitioning (sharding) and (2) that warehouses, orders and order lines are sharded by the warehouse id column, and that the elastic query can co-locate the joins on the shards and process the expensive part of the query on the shards in parallel.

```
select
 w_id as warehouse,
 o_c_id as customer,
 count(*) as cnt_orderline,
 max(ol_quantity) as max_quantity,
 avg(ol_amount) as avg_amount,
 min(ol_delivery_d) as min_deliv_date
from warehouse
join orders
on w_id = o_w_id
join order_line
on o_id = ol_o_id and o_w_id = ol_w_id
where w_id > 100 and w_id < 200
group by w_id, o_c_id
```

## Stored procedure for remote T-SQL execution: `sp_execute_remote`

Elastic query also introduces a stored procedure that provides direct access to the shards. The stored procedure is called `sp_execute_remote` and can be used to execute remote stored procedures or T-SQL code on the remote databases. It takes the following parameters:

- Data source name (nvarchar): The name of the external data source of type RDBMS.
- Query (nvarchar): The T-SQL query to be executed on each shard.
- Parameter declaration (nvarchar) - optional: String with data type definitions for the parameters used in the Query parameter (like `sp_executesql`).
- Parameter value list - optional: Comma-separated list of parameter values (like `sp_executesql`).

The `sp_execute_remote` uses the external data source provided in the invocation parameters to execute the given T-SQL statement on the remote databases. It uses the credential of the external data source to connect to the shardmap manager database and the remote databases.

Example:

```
EXEC sp_execute_remote
N'MyExtSrc',
N'select count(w_id) as foo from warehouse'
```

## Connectivity for tools

Use regular SQL Server connection strings to connect your application, your BI and data integration tools to the database with your external table definitions. Make sure that SQL Server is supported as a data source for your tool. Then reference the elastic query database like any other SQL Server database connected to the tool, and use external tables from your tool or application as if they were local tables.

## Best practices

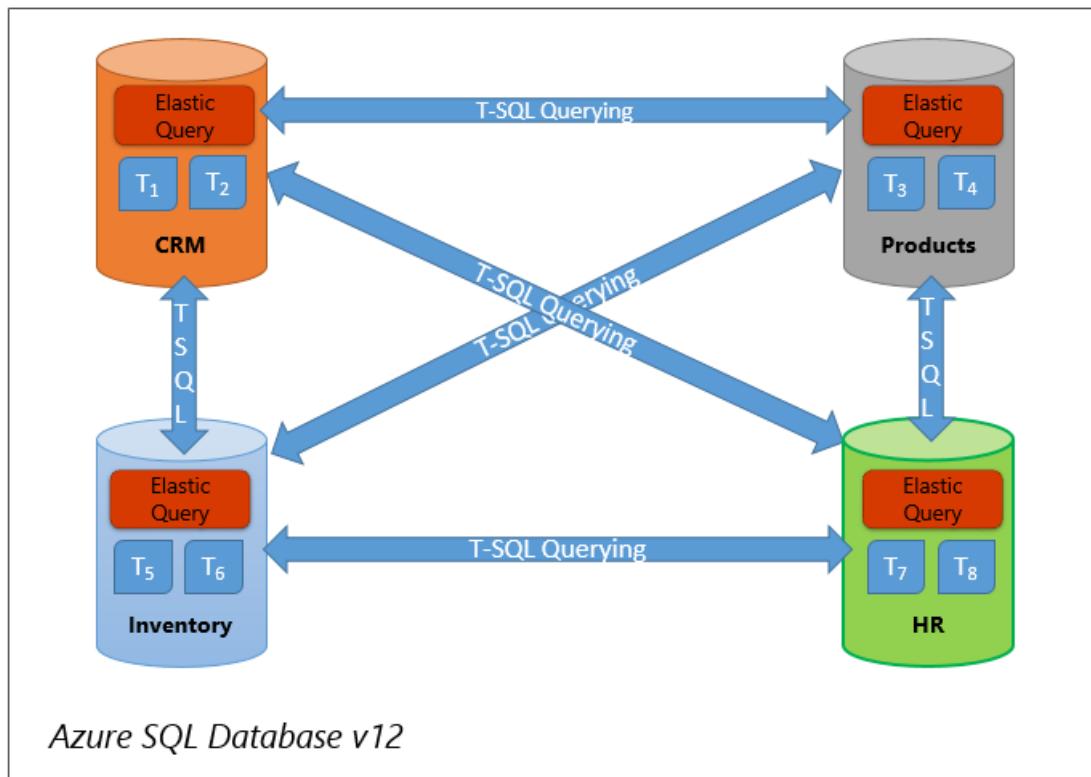
- Ensure that the elastic query endpoint database has been given access to the shardmap database and all shards through the SQL DB firewalls.
- Validate or enforce the data distribution defined by the external table. If your actual data distribution is different from the distribution specified in your table definition, your queries may yield unexpected results.
- Elastic query currently does not perform shard elimination when predicates over the sharding key would allow it to safely exclude certain shards from processing.
- Elastic query works best for queries where most of the computation can be done on the shards. You typically get the best query performance with selective filter predicates that can be evaluated on the shards or joins over the partitioning keys that can be performed in a partition-aligned way on all shards. Other query patterns may need to load large amounts of data from the shards to the head node and may perform poorly

## Next steps

- For an overview of elastic query, see [Elastic query overview](#).
- For a vertical partitioning tutorial, see [Getting started with cross-database query \(vertical partitioning\)](#).
- For syntax and sample queries for vertically partitioned data, see [Querying vertically partitioned data](#)
- For a horizontal partitioning (sharding) tutorial, see [Getting started with elastic query for horizontal partitioning \(sharding\)](#).
- See [sp\\_execute\\_remote](#) for a stored procedure that executes a Transact-SQL statement on a single remote Azure SQL Database or set of databases serving as shards in a horizontal partitioning scheme.

# Query across cloud databases with different schemas (preview)

11/7/2019 • 6 minutes to read • [Edit Online](#)



Vertically-partitioned databases use different sets of tables on different databases. That means that the schema is different on different databases. For instance, all tables for inventory are on one database while all accounting-related tables are on a second database.

## Prerequisites

- The user must possess ALTER ANY EXTERNAL DATA SOURCE permission. This permission is included with the ALTER DATABASE permission.
- ALTER ANY EXTERNAL DATA SOURCE permissions are needed to refer to the underlying data source.

## Overview

### NOTE

Unlike with horizontal partitioning, these DDL statements do not depend on defining a data tier with a shard map through the elastic database client library.

1. [CREATE MASTER KEY](#)
2. [CREATE DATABASE SCOPED CREDENTIAL](#)
3. [CREATE EXTERNAL DATA SOURCE](#)
4. [CREATE EXTERNAL TABLE](#)

# Create database scoped master key and credentials

The credential is used by the elastic query to connect to your remote databases.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'master_key_password';
CREATE DATABASE SCOPED CREDENTIAL <credential_name> WITH IDENTITY = '<username>',
SECRET = '<password>'
[];
```

## NOTE

Ensure that the `<username>` does not include any "`@servername`" suffix.

# Create external data sources

Syntax:

```
<External_Data_Source> ::=
CREATE EXTERNAL DATA SOURCE <data_source_name> WITH
 (TYPE = RDBMS,
 LOCATION = '<fully_qualified_server_name>',
 DATABASE_NAME = '<remote_database_name>',
 CREDENTIAL = <credential_name>
) [];
```

## IMPORTANT

The TYPE parameter must be set to **RDBMS**.

## Example

The following example illustrates the use of the CREATE statement for external data sources.

```
CREATE EXTERNAL DATA SOURCE RemoteReferenceData
WITH
(
 TYPE=RDBMS,
 LOCATION='myserver.database.windows.net',
 DATABASE_NAME='ReferenceData',
 CREDENTIAL= SqlUser
);
```

To retrieve the list of current external data sources:

```
select * from sys.external_data_sources;
```

## External Tables

Syntax:

```

CREATE EXTERNAL TABLE [database_name . [schema_name] . | schema_name .] table_name
({ <column_definition> } [,...n]
{ WITH (<rdbms_external_table_options>) }
)[;]

<rdbms_external_table_options> ::==
 DATA_SOURCE = <External_Data_Source>,
 [SCHEMA_NAME = N'nonescaped_schema_name',]
 [OBJECT_NAME = N'nonescaped_object_name',]

```

## Example

```

CREATE EXTERNAL TABLE [dbo].[customer](
 [c_id] int NOT NULL,
 [c_firstname] nvarchar(256) NULL,
 [c_lastname] nvarchar(256) NOT NULL,
 [street] nvarchar(256) NOT NULL,
 [city] nvarchar(256) NOT NULL,
 [state] nvarchar(20) NULL,
 [country] nvarchar(50) NOT NULL,
)
WITH
(
 DATA_SOURCE = RemoteReferenceData
);

```

The following example shows how to retrieve the list of external tables from the current database:

```
select * from sys.external_tables;
```

## Remarks

Elastic query extends the existing external table syntax to define external tables that use external data sources of type RDBMS. An external table definition for vertical partitioning covers the following aspects:

- **Schema:** The external table DDL defines a schema that your queries can use. The schema provided in your external table definition needs to match the schema of the tables in the remote database where the actual data is stored.
- **Remote database reference:** The external table DDL refers to an external data source. The external data source specifies the SQL Database server name and database name of the remote database where the actual table data is stored.

Using an external data source as outlined in the previous section, the syntax to create external tables is as follows:

The DATA\_SOURCE clause defines the external data source (i.e. the remote database in case of vertical partitioning) that is used for the external table.

The SCHEMA\_NAME and OBJECT\_NAME clauses provide the ability to map the external table definition to a table in a different schema on the remote database, or to a table with a different name, respectively. This is useful if you want to define an external table to a catalog view or DMV on your remote database - or any other situation where the remote table name is already taken locally.

The following DDL statement drops an existing external table definition from the local catalog. It does not impact the remote database.

```
DROP EXTERNAL TABLE [[schema_name] . | schema_name.] table_name[;]
```

**Permissions for CREATE/DROP EXTERNAL TABLE:** ALTER ANY EXTERNAL DATA SOURCE permissions

are needed for external table DDL which is also needed to refer to the underlying data source.

## Security considerations

Users with access to the external table automatically gain access to the underlying remote tables under the credential given in the external data source definition. You should carefully manage access to the external table in order to avoid undesired elevation of privileges through the credential of the external data source. Regular SQL permissions can be used to GRANT or REVOKE access to an external table just as though it were a regular table.

## Example: querying vertically partitioned databases

The following query performs a three-way join between the two local tables for orders and order lines and the remote table for customers. This is an example of the reference data use case for elastic query:

```
SELECT
 c_id as customer,
 c_lastname as customer_name,
 count(*) as cnt_orderline,
 max(ol_quantity) as max_quantity,
 avg(ol_amount) as avg_amount,
 min(ol_delivery_d) as min_deliv_date
FROM customer
JOIN orders
ON c_id = o_c_id
JOIN order_line
ON o_id = ol_o_id and o_c_id = ol_c_id
WHERE c_id = 100
```

## Stored procedure for remote T-SQL execution: sp\_execute\_remote

Elastic query also introduces a stored procedure that provides direct access to the remote database. The stored procedure is called `sp_execute_remote` and can be used to execute remote stored procedures or T-SQL code on the remote database. It takes the following parameters:

- Data source name (nvarchar): The name of the external data source of type RDBMS.
- Query (nvarchar): The T-SQL query to be executed on the remote database.
- Parameter declaration (nvarchar) - optional: String with data type definitions for the parameters used in the Query parameter (like `sp_executesql`).
- Parameter value list - optional: Comma-separated list of parameter values (like `sp_executesql`).

The `sp_execute_remote` uses the external data source provided in the invocation parameters to execute the given T-SQL statement on the remote database. It uses the credential of the external data source to connect to the remote database.

Example:

```
EXEC sp_execute_remote
 N'MyExtSrc',
 N'select count(w_id) as foo from warehouse'
```

## Connectivity for tools

You can use regular SQL Server connection strings to connect your BI and data integration tools to databases on the SQL DB server that has elastic query enabled and external tables defined. Make sure that SQL Server is supported as a data source for your tool. Then refer to the elastic query database and its external tables just like

any other SQL Server database that you would connect to with your tool.

## Best practices

- Ensure that the elastic query endpoint database has been given access to the remote database by enabling access for Azure Services in its SQL DB firewall configuration. Also ensure that the credential provided in the external data source definition can successfully log into the remote database and has the permissions to access the remote table.
- Elastic query works best for queries where most of the computation can be done on the remote databases. You typically get the best query performance with selective filter predicates that can be evaluated on the remote databases or joins that can be performed completely on the remote database. Other query patterns may need to load large amounts of data from the remote database and may perform poorly.

## Next steps

- For an overview of elastic query, see [Elastic query overview](#).
- For a vertical partitioning tutorial, see [Getting started with cross-database query \(vertical partitioning\)](#).
- For a horizontal partitioning (sharding) tutorial, see [Getting started with elastic query for horizontal partitioning \(sharding\)](#).
- For syntax and sample queries for horizontally partitioned data, see [Querying horizontally partitioned data](#)
- See [`sp\_execute\_remote`](#) for a stored procedure that executes a Transact-SQL statement on a single remote Azure SQL Database or set of databases serving as shards in a horizontal partitioning scheme.

# Create an Elastic Job agent using PowerShell

11/22/2019 • 8 minutes to read • [Edit Online](#)

[Elastic jobs](#) enable the running of one or more Transact-SQL (T-SQL) scripts in parallel across many databases.

In this tutorial, you learn the steps required to run a query across multiple databases:

- Create an Elastic Job agent
- Create job credentials so that jobs can execute scripts on its targets
- Define the targets (servers, elastic pools, databases, shard maps) you want to run the job against
- Create database scoped credentials in the target databases so the agent connect and execute jobs
- Create a job
- Add job steps to a job
- Start execution of a job
- Monitor a job

## Prerequisites

The upgraded version of Elastic Database jobs has a new set of PowerShell cmdlets for use during migration. These new cmdlets transfer all of your existing job credentials, targets (including databases, servers, custom collections), job triggers, job schedules, job contents, and jobs over to a new Elastic Job agent.

### Install the latest Elastic Jobs cmdlets

If you don't have already have an Azure subscription, [create a free account](#) before you begin.

Install the **Az.Sql** module to get the latest Elastic Job cmdlets. Run the following commands in PowerShell with administrative access.

```
installs the latest PackageManagement and PowerShellGet packages
Find-Package PackageManagement | Install-Package -Force
Find-Package PowerShellGet | Install-Package -Force

Restart your powershell session with administrative access

Install and import the Az.Sql module, then confirm
Install-Module -Name Az.Sql
Import-Module Az.Sql

Get-Module Az.Sql
```

In addition to the **Az.Sql** module, this tutorial also requires the *SqlServer* PowerShell module. For details, see [Install SQL Server PowerShell module](#).

## Create required resources

Creating an Elastic Job agent requires a database (S0 or higher) for use as the [Job database](#).

The script below creates a new resource group, server, and database for use as the Job database. The second script creates a second server with two blank databases to execute jobs against.

Elastic Jobs has no specific naming requirements so you can use whatever naming conventions you want, as long as they conform to any [Azure requirements](#).

```

sign in to Azure account
Connect-AzAccount

create a resource group
Write-Output "Creating a resource group..."
$resourceGroupName = Read-Host "Please enter a resource group name"
.setLocation = Read-Host "Please enter an Azure Region"
$rg = New-AzResourceGroup -Name $resourceGroupName -Location $location
$rg

create a server
Write-Output "Creating a server..."
$agentServerName = Read-Host "Please enter an agent server name"
$agentServerName = $agentServerName + "-" + [guid]::NewGuid()
$adminLogin = Read-Host "Please enter the server admin name"
$adminPassword = Read-Host "Please enter the server admin password"
$adminPasswordSecure = ConvertTo-SecureString -String $AdminPassword -AsPlainText -Force
$adminCred = New-Object -TypeName "System.Management.Automation.PSCredential" -ArgumentList $adminLogin,
$adminPasswordSecure
$agentServer = New-AzSqlServer -ResourceGroupName $resourceGroupName -Location $location `
-ServerName $agentServerName -ServerVersion "12.0" -SqlAdministratorCredentials ($adminCred)

set server firewall rules to allow all Azure IPs
Write-Output "Creating a server firewall rule..."
$agentServer | New-AzSqlServerFirewallRule -AllowAllAzureIPs
$agentServer

create the job database
Write-Output "Creating a blank SQL database to be used as the Job Database..."
$jobDatabaseName = "JobDatabase"
$jobDatabase = New-AzSqlDatabase -ResourceGroupName $resourceGroupName -ServerName $agentServerName -
DatabaseName $jobDatabaseName -RequestedServiceObjectiveName "S0"
$jobDatabase

```

```

create a target server and sample databases - uses the same credentials
Write-Output "Creating target server..."
$targetServerName = Read-Host "Please enter a target server name"
$targetServerName = $targetServerName + "-" + [guid]::NewGuid()
$targetServer = New-AzSqlServer -ResourceGroupName $resourceGroupName -Location $location `
-ServerName $targetServerName -ServerVersion "12.0" -SqlAdministratorCredentials ($adminCred)

set target server firewall rules to allow all Azure IPs
$targetServer | New-AzSqlServerFirewallRule -AllowAllAzureIPs
$targetServer | New-AzSqlServerFirewallRule -StartIpAddress 0.0.0.0 -EndIpAddress 255.255.255.255 -
FirewallRuleName AllowAll
$targetServer

create sample databases to execute jobs against
$db1 = New-AzSqlDatabase -ResourceGroupName $resourceGroupName -ServerName $targetServerName -DatabaseName
"database1"
$db1
$db2 = New-AzSqlDatabase -ResourceGroupName $resourceGroupName -ServerName $targetServerName -DatabaseName
"database2"
$db2

```

## Use Elastic Jobs

To use Elastic Jobs, register the feature in your Azure subscription by running the following command. Run this command once for the subscription in which you intend to provision the Elastic Job agent. Subscriptions that only contain databases that are job targets don't need to be registered.

```
Register-AzProviderFeature -FeatureName sqldb-JobAccounts -ProviderNamespace Microsoft.Sql
```

## Create the Elastic Job agent

An Elastic Job agent is an Azure resource for creating, running, and managing jobs. The agent executes jobs based on a schedule or as a one-time job.

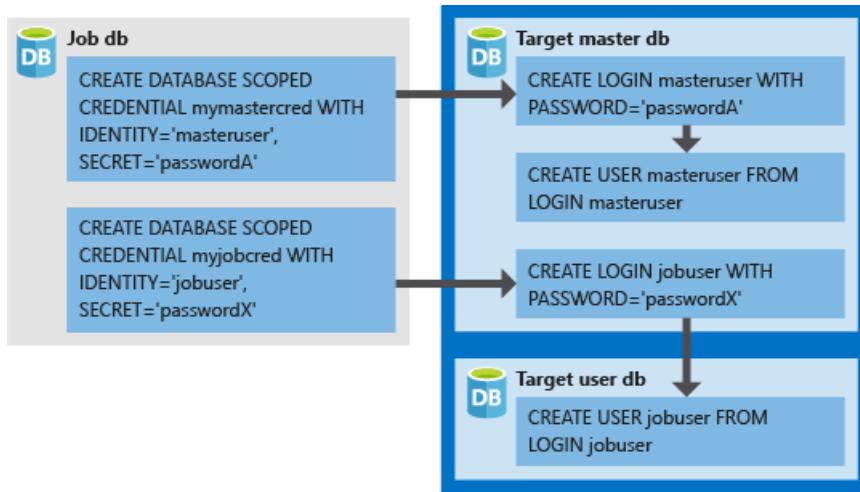
The **New-AzSqlElasticJobAgent** cmdlet requires an Azure SQL database to already exist, so the *resourceGroupName*, *serverName*, and *databaseName* parameters must all point to existing resources.

```
Write-Output "Creating job agent..."
$agentName = Read-Host "Please enter a name for your new Elastic Job agent"
$jobAgent = $jobDatabase | New-AzSqlElasticJobAgent -Name $agentName
$jobAgent
```

## Create the job credentials

Jobs use database scoped credentials to connect to the target databases specified by the target group upon execution and execute scripts. These database scoped credentials are also used to connect to the master database to enumerate all the databases in a server or an elastic pool, when either of these are used as the target group member type.

The database scoped credentials must be created in the job database. All target databases must have a login with sufficient permissions for the job to complete successfully.



In addition to the credentials in the image, note the addition of the **GRANT** commands in the following script. These permissions are required for the script we chose for this example job. Because the example creates a new table in the targeted databases, each target db needs the proper permissions to successfully run.

To create the required job credentials (in the job database), run the following script:

```

in the master database (target server)
create the master user login, master user, and job user login
$params = @{
 'database' = 'master'
 'serverInstance' = $targetServer.ServerName + '.database.windows.net'
 'username' = $adminLogin
 'password' = $adminPassword
 'outputSqlErrors' = $true
 'query' = "CREATE LOGIN masteruser WITH PASSWORD='password!123'"
}
Invoke-SqlCmd @params
$params.query = "CREATE USER masteruser FROM LOGIN masteruser"
Invoke-SqlCmd @params
$params.query = "CREATE LOGIN jobuser WITH PASSWORD='password!123'"
Invoke-SqlCmd @params

for each target database
create the jobuser from jobuser login and check permission for script execution
$targetDatabases = @($db1.DatabaseName, $Db2.DatabaseName)
$createJobUserScript = "CREATE USER jobuser FROM LOGIN jobuser"
$grantAlterSchemaScript = "GRANT ALTER ON SCHEMA::dbo TO jobuser"
$grantCreateScript = "GRANT CREATE TABLE TO jobuser"

$targetDatabases | % {
 $params.database = $_
 $params.query = $createJobUserScript
 Invoke-SqlCmd @params
 $params.query = $grantAlterSchemaScript
 Invoke-SqlCmd @params
 $params.query = $grantCreateScript
 Invoke-SqlCmd @params
}

create job credential in Job database for master user
Write-Output "Creating job credentials..."
$loginPasswordSecure = (ConvertTo-SecureString -String "password!123" -AsPlainText -Force)

$masterCred = New-Object -TypeName "System.Management.Automation.PSCredential" -ArgumentList "masteruser",
$loginPasswordSecure
$masterCred = $jobAgent | New-AzSqlElasticJobCredential -Name "masteruser" -Credential $masterCred

$jobCred = New-Object -TypeName "System.Management.Automation.PSCredential" -ArgumentList "jobuser",
$loginPasswordSecure
$jobCred = $jobAgent | New-AzSqlElasticJobCredential -Name "jobuser" -Credential $jobCred

```

## Define the target databases to run the job against

A [target group](#) defines the set of one or more databases a job step will execute on.

The following snippet creates two target groups: `serverGroup`, and `serverGroupExcludingDb2`. `serverGroup` targets all databases that exist on the server at the time of execution, and `serverGroupExcludingDb2` targets all databases on the server, except `targetDb2`:

```

Write-Output "Creating test target groups..."
create ServerGroup target group
$serverGroup = $jobAgent | New-AzSqlElasticJobTargetGroup -Name 'ServerGroup'
$serverGroup | Add-AzSqlElasticJobTarget -ServerName $targetServerName -RefreshCredentialName
$masterCred.CredentialName

create ServerGroup with an exclusion of db2
$serverGroupExcludingDb2 = $jobAgent | New-AzSqlElasticJobTargetGroup -Name 'ServerGroupExcludingDb2'
$serverGroupExcludingDb2 | Add-AzSqlElasticJobTarget -ServerName $targetServerName -RefreshCredentialName
$masterCred.CredentialName
$serverGroupExcludingDb2 | Add-AzSqlElasticJobTarget -ServerName $targetServerName -Database
$db2.DatabaseName -Exclude

```

## Create a job and steps

This example defines a job and two job steps for the job to run. The first job step (*step1*) creates a new table (*Step1Table*) in every database in target group *ServerGroup*. The second job step (*step2*) creates a new table (*Step2Table*) in every database except for *TargetDb2*, because the target group defined previously specified to exclude it.

```

Write-Output "Creating a new job..."
$jobName = "Job1"
$job = $jobAgent | New-AzSqlElasticJob -Name $jobName -RunOnce
$job

Write-Output "Creating job steps..."
$sqlText1 = "IF NOT EXISTS (SELECT * FROM sys.tables WHERE object_id = object_id('Step1Table')) CREATE TABLE
[dbo].[Step1Table]([TestId] [int] NOT NULL);"
$sqlText2 = "IF NOT EXISTS (SELECT * FROM sys.tables WHERE object_id = object_id('Step2Table')) CREATE TABLE
[dbo].[Step2Table]([TestId] [int] NOT NULL);"

$job | Add-AzSqlElasticJobStep -Name "step1" -TargetGroupName $serverGroup.TargetGroupName -CredentialName
$jobCred.CredentialName -CommandText $sqlText1
$job | Add-AzSqlElasticJobStep -Name "step2" -TargetGroupName $serverGroupExcludingDb2.TargetGroupName -
CredentialName $jobCred.CredentialName -CommandText $sqlText2

```

## Run the job

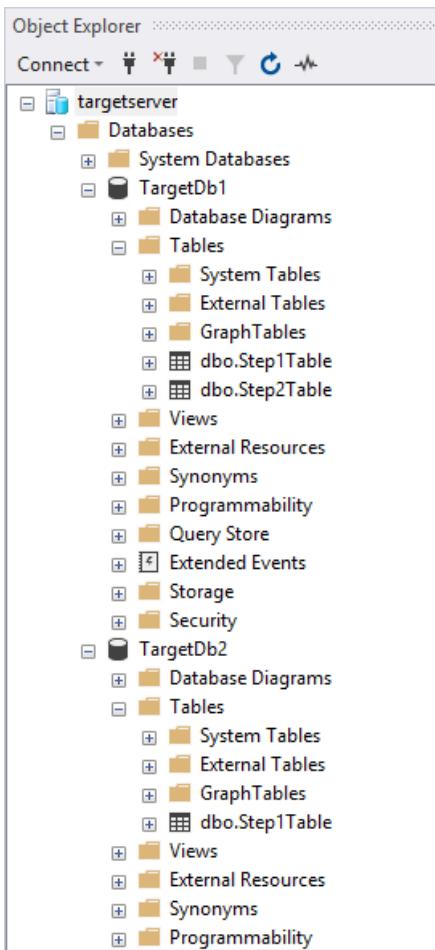
To start the job immediately, run the following command:

```

Write-Output "Start a new execution of the job..."
$jobExecution = $job | Start-AzSqlElasticJob
$jobExecution

```

After successful completion you should see two new tables in *TargetDb1*, and only one new table in *TargetDb2*:



You can also schedule the job to run later. To schedule a job to run at a specific time, run the following command:

```
run every hour starting from now
$job | Set-AzSqlElasticJob -IntervalType Hour -IntervalCount 1 -StartTime (Get-Date) -Enable
```

## Monitor status of job executions

The following snippets get job execution details:

```
get the latest 10 executions run
$jobAgent | Get-AzSqlElasticJobExecution -Count 10

get the job step execution details
$jobExecution | Get-AzSqlElasticJobStepExecution

get the job target execution details
$jobExecution | Get-AzSqlElasticJobTargetExecution -Count 2
```

The following table lists the possible job execution states:

STATE	DESCRIPTION
<b>Created</b>	The job execution was just created and is not yet in progress.
<b>InProgress</b>	The job execution is currently in progress.
<b>WaitingForRetry</b>	The job execution wasn't able to complete its action and is waiting to retry.

STATE	DESCRIPTION
<b>Succeeded</b>	The job execution has completed successfully.
<b>SucceededWithSkipped</b>	The job execution has completed successfully, but some of its children were skipped.
<b>Failed</b>	The job execution has failed and exhausted its retries.
<b>TimedOut</b>	The job execution has timed out.
<b>Canceled</b>	The job execution was canceled.
<b>Skipped</b>	The job execution was skipped because another execution of the same job step was already running on the same target.
<b>WaitingForChildJobExecutions</b>	The job execution is waiting for its child executions to complete.

## Clean up resources

Delete the Azure resources created in this tutorial by deleting the resource group.

### TIP

If you plan to continue to work with these jobs, you do not clean up the resources created in this article.

```
Remove-AzResourceGroup -ResourceGroupName $resourceGroupName
```

## Next steps

In this tutorial, you ran a Transact-SQL script against a set of databases. You learned how to do the following tasks:

- Create an Elastic Job agent
- Create job credentials so that jobs can execute scripts on its targets
- Define the targets (servers, elastic pools, databases, shard maps) you want to run the job against
- Create database scoped credentials in the target databases so the agent connect and execute jobs
- Create a job
- Add a job step to the job
- Start an execution of the job
- Monitor the job

[Manage Elastic Jobs using Transact-SQL](#)

# Use Transact-SQL (T-SQL) to create and manage Elastic Database Jobs

2/7/2020 • 40 minutes to read • [Edit Online](#)

This article provides many example scenarios to get started working with Elastic Jobs using T-SQL.

The examples use the [stored procedures](#) and [views](#) available in the [job database](#).

Transact-SQL (T-SQL) is used to create, configure, execute, and manage jobs. Creating the Elastic Job agent is not supported in T-SQL, so you must first create an *Elastic Job agent* using the portal, or [PowerShell](#).

## Create a credential for job execution

The credential is used to connect to your target databases for script execution. The credential needs appropriate permissions, on the databases specified by the target group, to successfully execute the script. When using a server and/or pool target group member, it is highly suggested to create a master credential for use to refresh the credential prior to expansion of the server and/or pool at time of job execution. The database scoped credential is created on the job agent database. The same credential must be used to *Create a Login* and *Create a User from Login to grant the Login Database Permissions* on the target databases.

```
--Connect to the job database specified when creating the job agent

-- Create a db master key if one does not already exist, using your own password.
CREATE MASTER KEY ENCRYPTION BY PASSWORD='<EnterStrongPasswordHere>';

-- Create a database scoped credential.
CREATE DATABASE SCOPED CREDENTIAL myjobcred WITH IDENTITY = 'jobcred',
 SECRET = '<EnterStrongPasswordHere>';
GO

-- Create a database scoped credential for the master database of server1.
CREATE DATABASE SCOPED CREDENTIAL mymastercred WITH IDENTITY = 'mastercred',
 SECRET = '<EnterStrongPasswordHere>';
GO
```

## Create a target group (servers)

The following example shows how to execute a job against all databases in a server.

Connect to the [job database](#) and run the following command:

```
-- Connect to the job database specified when creating the job agent

-- Add a target group containing server(s)
EXEC jobs.sp_add_target_group 'ServerGroup1'

-- Add a server target member
EXEC jobs.sp_add_target_group_member
'ServerGroup1',
@target_type = 'SqlServer',
@refreshCredential_name='mymastercred', --credential required to refresh the databases in server
@server_name='server1.database.windows.net'

--View the recently created target group and target group members
SELECT * FROM jobs.target_groups WHERE target_group_name='ServerGroup1';
SELECT * FROM jobs.target_group_members WHERE target_group_name='ServerGroup1';
```

## Exclude an individual database

The following example shows how to execute a job against all databases in a SQL Database server, except for the database named *MappingDB*.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

-- Add a target group containing server(s)
EXEC [jobs].sp_add_target_group N'ServerGroup'
GO

-- Add a server target member
EXEC [jobs].sp_add_target_group_member
@target_group_name = N'ServerGroup',
@target_type = N'SqlServer',
@refreshCredential_name=N'mymastercred', --credential required to refresh the databases in server
@server_name=N'London.database.windows.net'
GO

-- Add a server target member
EXEC [jobs].sp_add_target_group_member
@target_group_name = N'ServerGroup',
@target_type = N'SqlServer',
@refreshCredential_name=N'mymastercred', --credential required to refresh the databases in server
@server_name='server2.database.windows.net'
GO

--Exclude a database target member from the server target group
EXEC [jobs].sp_add_target_group_member
@target_group_name = N'ServerGroup',
@membership_type = N'Exclude',
@target_type = N'SqlDatabase',
@server_name = N'server1.database.windows.net',
@database_name =N'MappingDB'
GO

--View the recently created target group and target group members
SELECT * FROM [jobs].target_groups WHERE target_group_name = N'ServerGroup';
SELECT * FROM [jobs].target_group_members WHERE target_group_name = N'ServerGroup';
```

## Create a target group (pools)

The following example shows how to target all the databases in one or more elastic pools.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

-- Add a target group containing pool(s)
EXEC jobs.sp_add_target_group 'PoolGroup'

-- Add an elastic pool(s) target member
EXEC jobs.sp_add_target_group_member
'PoolGroup',
@target_type = 'SqlElasticPool',
@refresh_credential_name='mymastercred', --credential required to refresh the databases in server
@server_name='server1.database.windows.net',
@elastic_pool_name='ElasticPool-1'

-- View the recently created target group and target group members
SELECT * FROM jobs.target_groups WHERE target_group_name = N'PoolGroup';
SELECT * FROM jobs.target_group_members WHERE target_group_name = N'PoolGroup';
```

## Deploy new schema to many databases

The following example shows how to deploy new schema to all databases.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

--Add job for create table
EXEC jobs.sp_add_job @job_name='CreateTableTest', @description='Create Table Test'

-- Add job step for create table
EXEC jobs.sp_add_jobstep @job_name='CreateTableTest',
@command=N'IF NOT EXISTS (SELECT * FROM sys.tables
 WHERE object_id = object_id(''Test''))
CREATE TABLE [dbo].[Test]([TestId] [int] NOT NULL);',
@credential_name='myjobcred',
@target_group_name='PoolGroup'
```

## Data collection using built-in parameters

In many data collection scenarios, it can be useful to include some of these scripting variables to help post-process the results of the job.

- \$(job\_name)
- \$(job\_id)
- \$(job\_version)
- \$(step\_id)
- \$(step\_name)
- \$(job\_execution\_id)
- \$(job\_execution\_create\_time)
- \$(target\_group\_name)

For example, to group all results from the same job execution together, use the `$(job_execution_id)` as shown in the following command:

```
@command= N' SELECT DB_NAME() DatabaseName, $(job_execution_id) AS job_execution_id, * FROM
sys.dm_db_resource_stats WHERE end_time > DATEADD(mi, -20, GETDATE());'
```

## Monitor database performance

The following example creates a new job to collect performance data from multiple databases.

By default, the job agent will create the output table to store returned results. Therefore, the database principal associated with the output credential must at a minimum have the following permissions: `CREATE TABLE` on the database, `ALTER`, `SELECT`, `INSERT`, `DELETE` on the output table or its schema, and `SELECT` on the `sys.indexes` catalog view.

If you want to manually create the table ahead of time then it needs to have the following properties:

1. Columns with the correct name and data types for the result set.
2. Additional column for `internal_execution_id` with the data type of `uniqueidentifier`.
3. A nonclustered index named `IX_<TableName>_Internal_Execution_ID` on the `internal_execution_id` column.
4. All permissions listed above except for `CREATE TABLE` permission on the database.

Connect to the *job database* and run the following commands:

```

--Connect to the job database specified when creating the job agent

-- Add a job to collect perf results
EXEC jobs.sp_add_job @job_name = 'ResultsJob', @description='Collection Performance data from all customers'

-- Add a job step w/ schedule to collect results
EXEC jobs.sp_add_jobstep
@job_name='ResultsJob',
@command= N' SELECT DB_NAME() DatabaseName, $(job_execution_id) AS job_execution_id, * FROM
sys.dm_db_resource_stats WHERE end_time > DATEADD(mi, -20, GETDATE());',
@credential_name='myjobcred',
@target_group_name='PoolGroup',
@output_type='SqlDatabase',
@output_credential_name='myjobcred',
@output_server_name='server1.database.windows.net',
@output_database_name='<resultsdb>',
@output_table_name='<resulstable>'

Create a job to monitor pool performance
--Connect to the job database specified when creating the job agent

-- Add a target group containing master database
EXEC jobs.sp_add_target_group 'MasterGroup'

-- Add a server target member
EXEC jobs.sp_add_target_group_member
@target_group_name='MasterGroup',
@target_type='SqlDatabase',
@server_name='server1.database.windows.net',
@database_name='master'

-- Add a job to collect perf results
EXEC jobs.sp_add_job
@job_name='ResultsPoolsJob',
@description='Demo: Collection Performance data from all pools',
@schedule_interval_type='Minutes',
@schedule_interval_count=15

-- Add a job step w/ schedule to collect results
EXEC jobs.sp_add_jobstep
@job_name='ResultsPoolsJob',
@command=N'declare @now datetime
DECLARE @startTime datetime
DECLARE @endTime datetime
DECLARE @poolLagMinutes datetime
DECLARE @poolStartTime datetime
DECLARE @poolEndTime datetime
SELECT @now = getutcdate ()
SELECT @startTime = dateadd(minute, -15, @now)
SELECT @endTime = @now
SELECT @poolStartTime = dateadd(minute, -30, @startTime)
SELECT @poolEndTime = dateadd(minute, -30, @endTime)

SELECT elastic_pool_name , end_time, elastic_pool_dtu_limit, avg_cpu_percent, avg_data_io_percent,
avg_log_write_percent, max_worker_percent, max_session_percent,
avg_storage_percent, elastic_pool_storage_limit_mb FROM sys.elastic_pool_resource_stats
WHERE end_time > @poolStartTime and end_time <= @poolEndTime;
',

@credential_name='myjobcred',
@target_group_name='MasterGroup',
@output_type='SqlDatabase',
@output_credential_name='myjobcred',
@output_server_name='server1.database.windows.net',
@output_database_name='resultsdb',
@output_table_name='resulstable'

```

## View job definitions

The following example shows how to view current job definitions.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

-- View all jobs
SELECT * FROM jobs.jobs

-- View the steps of the current version of all jobs
SELECT js.* FROM jobs.jobsteps js
JOIN jobs.jobs j
 ON j.job_id = js.job_id AND j.job_version = js.job_version

-- View the steps of all versions of all jobs
select * from jobs.jobsteps
```

## Begin ad hoc execution of a job

The following example shows how to start a job immediately.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

-- Execute the latest version of a job
EXEC jobs.sp_start_job 'CreateTableTest'

-- Execute the latest version of a job and receive the execution id
declare @je uniqueidentifier
exec jobs.sp_start_job 'CreateTableTest', @job_execution_id = @je output
select @je

select * from jobs.job_executions where job_execution_id = @je

-- Execute a specific version of a job (e.g. version 1)
exec jobs.sp_start_job 'CreateTableTest', 1
```

## Schedule execution of a job

The following example shows how to schedule a job for future execution.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

EXEC jobs.sp_update_job
@job_name='ResultsJob',
@enabled=1,
@schedule_interval_type='Minutes',
@schedule_interval_count=15
```

## Monitor job execution status

The following example shows how to view execution status details for all jobs.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

--View top-level execution status for the job named 'ResultsPoolJob'
SELECT * FROM jobs.job_executions
WHERE job_name = 'ResultsPoolsJob' and step_id IS NULL
ORDER BY start_time DESC

--View all top-level execution status for all jobs
SELECT * FROM jobs.job_executions WHERE step_id IS NULL
ORDER BY start_time DESC

--View all execution statuses for job named 'ResultsPoolsJob'
SELECT * FROM jobs.job_executions
WHERE job_name = 'ResultsPoolsJob'
ORDER BY start_time DESC

-- View all active executions
SELECT * FROM jobs.job_executions
WHERE is_active = 1
ORDER BY start_time DESC
```

## Cancel a job

The following example shows how to cancel a job.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

-- View all active executions to determine job execution id
SELECT * FROM jobs.job_executions
WHERE is_active = 1 AND job_name = 'ResultPoolsJob'
ORDER BY start_time DESC
GO

-- Cancel job execution with the specified job execution id
EXEC jobs.sp_stop_job '01234567-89ab-cdef-0123-456789abcdef'
```

## Delete old job history

The following example shows how to delete job history prior to a specific date.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent

-- Delete history of a specific job's executions older than the specified date
EXEC jobs.sp_purge_jobhistory @job_name='ResultPoolsJob', @oldest_date='2016-07-01 00:00:00'

--Note: job history is automatically deleted if it is >45 days old
```

## Delete a job and all its job history

The following example shows how to delete a job and all related job history.

Connect to the [job database](#) and run the following command:

```
--Connect to the job database specified when creating the job agent
```

```
EXEC jobs.sp_delete_job @job_name='ResultsPoolsJob'
```

```
--Note: job history is automatically deleted if it is >45 days old
```

## Job stored procedures

The following stored procedures are in the [jobs database](#).

STORED PROCEDURE	DESCRIPTION
<a href="#">sp_add_job</a>	Adds a new job.
<a href="#">sp_update_job</a>	Updates an existing job.
<a href="#">sp_delete_job</a>	Deletes an existing job.
<a href="#">sp_add_jobstep</a>	Adds a step to a job.
<a href="#">sp_update_jobstep</a>	Updates a job step.
<a href="#">sp_delete_jobstep</a>	Deletes a job step.
<a href="#">sp_start_job</a>	Starts executing a job.
<a href="#">sp_stop_job</a>	Stops a job execution.
<a href="#">sp_add_target_group</a>	Adds a target group.
<a href="#">sp_delete_target_group</a>	Deletes a target group.
<a href="#">sp_add_target_group_member</a>	Adds a database or group of databases to a target group.
<a href="#">sp_delete_target_group_member</a>	Removes a target group member from a target group.
<a href="#">sp_purge_jobhistory</a>	Removes the history records for a job.

### **sp\_add\_job**

Adds a new job.

#### Syntax

```
[jobs].sp_add_job [@job_name =] 'job_name'
[, [@description =] 'description']
[, [@enabled =] enabled]
[, [@schedule_interval_type =] schedule_interval_type]
[, [@schedule_interval_count =] schedule_interval_count]
[, [@schedule_start_time =] schedule_start_time]
[, [@schedule_end_time =] schedule_end_time]
[, [@job_id =] job_id OUTPUT]
```

#### Arguments

[ **@job\_name** = ] 'job\_name'

The name of the job. The name must be unique and cannot contain the percent (%) character. job\_name is nvarchar(128), with no default.

[ **@description** = ] 'description'

The description of the job. description is nvarchar(512), with a default of NULL. If description is omitted, an empty string is used.

[ **@enabled** = ] enabled

Whether the job's schedule is enabled. Enabled is bit, with a default of 0 (disabled). If 0, the job is not enabled and does not run according to its schedule; however, it can be run manually. If 1, the job will run according to its schedule, and can also be run manually.

[ **@schedule\_interval\_type** = ] schedule\_interval\_type

Value indicates when the job is to be executed. schedule\_interval\_type is nvarchar(50), with a default of Once, and can be one of the following values:

- 'Once',
- 'Minutes',
- 'Hours',
- 'Days',
- 'Weeks',
- 'Months'

[ **@schedule\_interval\_count** = ] schedule\_interval\_count

Number of schedule\_interval\_count periods to occur between each execution of the job. schedule\_interval\_count is int, with a default of 1. The value must be greater than or equal to 1.

[ **@schedule\_start\_time** = ] schedule\_start\_time

Date on which job execution can begin. schedule\_start\_time is DATETIME2, with the default of 0001-01-01 00:00:00.0000000.

[ **@schedule\_end\_time** = ] schedule\_end\_time

Date on which job execution can stop. schedule\_end\_time is DATETIME2, with the default of 9999-12-31 11:59:59.0000000.

[ **@job\_id** = ] job\_id OUTPUT

The job identification number assigned to the job if created successfully. job\_id is an output variable of type uniqueidentifier.

#### Return Code Values

0 (success) or 1 (failure)

#### Remarks

sp\_add\_job must be run from the job agent database specified when creating the job agent. After sp\_add\_job has been executed to add a job, sp\_add\_jobstep can be used to add steps that perform the activities for the job. The job's initial version number is 0, which will be incremented to 1 when the first step is added.

#### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

## sp\_update\_job

Updates an existing job.

## Syntax

```
[jobs].sp_update_job [@job_name =] 'job_name'
[, [@new_name =] 'new_name']
[, [@description =] 'description']
[, [@enabled =] enabled]
[, [@schedule_interval_type =] schedule_interval_type]
[, [@schedule_interval_count =] schedule_interval_count]
[, [@schedule_start_time =] schedule_start_time]
[, [@schedule_end_time =] schedule_end_time]
```

## Arguments

**[ @job\_name = ] 'job\_name'**

The name of the job to be updated. job\_name is nvarchar(128).

**[ @new\_name = ] 'new\_name'**

The new name of the job. new\_name is nvarchar(128).

**[ @description = ] 'description'**

The description of the job. description is nvarchar(512).

**[ @enabled = ] enabled**

Specifies whether the job's schedule is enabled (1) or not enabled (0). Enabled is bit.

**[ @schedule\_interval\_type= ] schedule\_interval\_type**

Value indicates when the job is to be executed. schedule\_interval\_type is nvarchar(50) and can be one of the following values:

- 'Once',
- 'Minutes',
- 'Hours',
- 'Days',
- 'Weeks',
- 'Months'

**[ @schedule\_interval\_count= ] schedule\_interval\_count**

Number of schedule\_interval\_count periods to occur between each execution of the job. schedule\_interval\_count is int, with a default of 1. The value must be greater than or equal to 1.

**[ @schedule\_start\_time= ] schedule\_start\_time**

Date on which job execution can begin. schedule\_start\_time is DATETIME2, with the default of 0001-01-01 00:00:00.0000000.

**[ @schedule\_end\_time= ] schedule\_end\_time**

Date on which job execution can stop. schedule\_end\_time is DATETIME2, with the default of 9999-12-31 11:59:59.0000000.

## Return Code Values

0 (success) or 1 (failure)

## Remarks

After sp\_add\_job has been executed to add a job, sp\_add\_jobstep can be used to add steps that perform the activities for the job. The job's initial version number is 0, which will be incremented to 1 when the first step is added.

## Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to

just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### **sp\_delete\_job**

Deletes an existing job.

#### **Syntax**

```
[jobs].sp_delete_job [@job_name =] 'job_name'
[, [@force =] force]
```

#### **Arguments**

[ **@job\_name** = ] 'job\_name'

The name of the job to be deleted. job\_name is nvarchar(128).

[ **@force** = ] force

Specifies whether to delete if the job has any executions in progress and cancel all in-progress executions (1) or fail if any job executions are in progress (0). force is bit.

#### **Return Code Values**

0 (success) or 1 (failure)

#### **Remarks**

Job history is automatically deleted when a job is deleted.

#### **Permissions**

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### **sp\_add\_jobstep**

Adds a step to a job.

#### **Syntax**

```
[jobs].sp_add_jobstep [@job_name =] 'job_name'
[, [@step_id =] step_id]
[, [@step_name =] step_name]
[, [@command_type =] 'command_type']
[, [@command_source =] 'command_source']
, [@command =] 'command'
, [@credential_name =] 'credential_name'
, [@target_group_name =] 'target_group_name'
[, [@initial_retry_interval_seconds =] initial_retry_interval_seconds]
[, [@maximum_retry_interval_seconds =] maximum_retry_interval_seconds]
[, [@retry_interval_backoff_multiplier =] retry_interval_backoff_multiplier]
[, [@retry_attempts =] retry_attempts]
[, [@step_timeout_seconds =] step_timeout_seconds]
[, [@output_type =] 'output_type']
[, [@output_credential_name =] 'output_credential_name']
[, [@output_subscription_id =] 'output_subscription_id']
[, [@output_resource_group_name =] 'output_resource_group_name']
[, [@output_server_name =] 'output_server_name']
[, [@output_database_name =] 'output_database_name']
[, [@output_schema_name =] 'output_schema_name']
[, [@output_table_name =] 'output_table_name']
[, [@job_version =] job_version OUTPUT]
[, [@max_parallelism =] max_parallelism]
```

## Arguments

[ **@job\_name** = ] 'job\_name'

The name of the job to which to add the step. job\_name is nvarchar(128).

[ **@step\_id** = ] step\_id

The sequence identification number for the job step. Step identification numbers start at 1 and increment without gaps. If an existing step already has this id, then that step and all following steps will have their id's incremented so that this new step can be inserted into the sequence. If not specified, the step\_id will be automatically assigned to the last in the sequence of steps. step\_id is an int.

[ **@step\_name** = ] step\_name

The name of the step. Must be specified, except for the first step of a job which (for convenience) has a default name of 'JobStep'. step\_name is nvarchar(128).

[ **@command\_type** = ] 'command\_type'

The type of command that is executed by this jobstep. command\_type is nvarchar(50), with a default value of TSql, meaning that the value of the @command\_type parameter is a T-SQL script.

If specified, the value must be TSql.

[ **@command\_source** = ] 'command\_source'

The type of location where the command is stored. command\_source is nvarchar(50), with a default value of Inline, meaning that the value of the @command\_source parameter is the literal text of the command.

If specified, the value must be Inline.

[ **@command** = ] 'command'

The command must be valid T-SQL script and is then executed by this job step. command is nvarchar(max), with a default of NULL.

[ **@credential\_name** = ] 'credential\_name'

The name of the database scoped credential stored in this job control database that is used to connect to each of the target databases within the target group when this step is executed. credential\_name is nvarchar(128).

[ **@target\_group\_name** = ] 'target-group\_name'

The name of the target group that contains the target databases that the job step will be executed on.

target\_group\_name is nvarchar(128).

[ **@initial\_retry\_interval\_seconds** = ] initial\_retry\_interval\_seconds

The delay before the first retry attempt, if the job step fails on the initial execution attempt.

initial\_retry\_interval\_seconds is int, with default value of 1.

[ **@maximum\_retry\_interval\_seconds** = ] maximum\_retry\_interval\_seconds

The maximum delay between retry attempts. If the delay between retries would grow larger than this value, it is capped to this value instead. maximum\_retry\_interval\_seconds is int, with default value of 120.

[ **@retry\_interval\_backoff\_multiplier** = ] retry\_interval\_backoff\_multiplier

The multiplier to apply to the retry delay if multiple job step execution attempts fail. For example, if the first retry had a delay of 5 second and the backoff multiplier is 2.0, then the second retry will have a delay of 10 seconds and the third retry will have a delay of 20 seconds. retry\_interval\_backoff\_multiplier is real, with default value of 2.0.

[ **@retry\_attempts** = ] retry\_attempts

The number of times to retry execution if the initial attempt fails. For example, if the retry\_attempts value is 10, then there will be 1 initial attempt and 10 retry attempts, giving a total of 11 attempts. If the final retry attempt fails, then the job execution will terminate with a lifecycle of Failed. retry\_attempts is int, with default value of 10.

[ **@step\_timeout\_seconds** = ] step\_timeout\_seconds

The maximum amount of time allowed for the step to execute. If this time is exceeded, then the job execution will terminate with a lifecycle of TimedOut. step\_timeout\_seconds is int, with default value of 43,200 seconds (12 hours).

[ **@output\_type** = ] 'output\_type'

If not null, the type of destination that the command's first result set is written to. output\_type is nvarchar(50), with a default of NULL.

If specified, the value must be SqlDatabase.

[ **@output\_credential\_name** = ] 'output\_credential\_name'

If not null, the name of the database scoped credential that is used to connect to the output destination database.

Must be specified if output\_type equals SqlDatabase. output\_credential\_name is nvarchar(128), with a default value of NULL.

[ **@output\_subscription\_id** = ] 'output\_subscription\_id'

Needs description.

[ **@output\_resource\_group\_name** = ] 'output\_resource\_group\_name'

Needs description.

[ **@output\_server\_name** = ] 'output\_server\_name'

If not null, the fully qualified DNS name of the server that contains the output destination database. Must be specified if output\_type equals SqlDatabase. output\_server\_name is nvarchar(256), with a default of NULL.

[ **@output\_database\_name** = ] 'output\_database\_name'

If not null, the name of the database that contains the output destination table. Must be specified if output\_type equals SqlDatabase. output\_database\_name is nvarchar(128), with a default of NULL.

[ **@output\_schema\_name** = ] 'output\_schema\_name'

If not null, the name of the SQL schema that contains the output destination table. If output\_type equals SqlDatabase, the default value is dbo. output\_schema\_name is nvarchar(128).

[ **@output\_table\_name** = ] 'output\_table\_name'

If not null, the name of the table that the command's first result set will be written to. If the table doesn't already exist, it will be created based on the schema of the returning result-set. Must be specified if output\_type equals SqlDatabase. output\_table\_name is nvarchar(128), with a default value of NULL.

[ @job\_version = ] job\_version OUTPUT

Output parameter that will be assigned the new job version number. job\_version is int.

[ @max\_parallelism = ] max\_parallelism OUTPUT

The maximum level of parallelism per elastic pool. If set, then the job step will be restricted to only run on a maximum of that many databases per elastic pool. This applies to each elastic pool that is either directly included in the target group or is inside a server that is included in the target group. max\_parallelism is int.

#### Return Code Values

0 (success) or 1 (failure)

#### Remarks

When sp\_add\_jobstep succeeds, the job's current version number is incremented. The next time the job is executed, the new version will be used. If the job is currently executing, that execution will not contain the new step.

#### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### sp\_update\_jobstep

Updates a job step.

#### Syntax

```
[jobs].sp_update_jobstep [@job_name =] 'job_name'
[, [@step_id =] step_id]
[, [@step_name =] 'step_name']
[, [@new_id =] new_id]
[, [@new_name =] 'new_name']
[, [@command_type =] 'command_type']
[, [@command_source =] 'command_source']
[, [@command =] 'command'
, [@credential_name =] 'credential_name'
, [@target_group_name =] 'target_group_name'
[, [@initial_retry_interval_seconds =] initial_retry_interval_seconds]
[, [@maximum_retry_interval_seconds =] maximum_retry_interval_seconds]
[, [@retry_interval_backoff_multiplier =] retry_interval_backoff_multiplier]
[, [@retry_attempts =] retry_attempts]
[, [@step_timeout_seconds =] step_timeout_seconds]
[, [@output_type =] 'output_type']
[, [@output_credential_name =] 'output_credential_name']
[, [@output_server_name =] 'output_server_name']
[, [@output_database_name =] 'output_database_name']
[, [@output_schema_name =] 'output_schema_name']
[, [@output_table_name =] 'output_table_name']
[, [@job_version =] job_version OUTPUT]
[, [@max_parallelism =] max_parallelism]
```

#### Arguments

[ @job\_name = ] 'job\_name'

The name of the job to which the step belongs. job\_name is nvarchar(128).

[ @step\_id = ] step\_id

The identification number for the job step to be modified. Either step\_id or step\_name must be specified. step\_id is

an int.

[ **@step\_name** = ] 'step\_name'

The name of the step to be modified. Either step\_id or step\_name must be specified. step\_name is nvarchar(128).

[ **@new\_id** = ] new\_id

The new sequence identification number for the job step. Step identification numbers start at 1 and increment without gaps. If a step is reordered, then other steps will be automatically renumbered.

[ **@new\_name** = ] 'new\_name'

The new name of the step. new\_name is nvarchar(128).

[ **@command\_type** = ] 'command\_type'

The type of command that is executed by this jobstep. command\_type is nvarchar(50), with a default value of TSql, meaning that the value of the @command\_type parameter is a T-SQL script.

If specified, the value must be TSql.

[ **@command\_source** = ] 'command\_source'

The type of location where the command is stored. command\_source is nvarchar(50), with a default value of Inline, meaning that the value of the @command\_source parameter is the literal text of the command.

If specified, the value must be Inline.

[ **@command** = ] 'command'

The command(s) must be valid T-SQL script and is then executed by this job step. command is nvarchar(max), with a default of NULL.

[ **@credential\_name** = ] 'credential\_name'

The name of the database scoped credential stored in this job control database that is used to connect to each of the target databases within the target group when this step is executed. credential\_name is nvarchar(128).

[ **@target\_group\_name** = ] 'target-group\_name'

The name of the target group that contains the target databases that the job step will be executed on. target\_group\_name is nvarchar(128).

[ **@initial\_retry\_interval\_seconds** = ] initial\_retry\_interval\_seconds

The delay before the first retry attempt, if the job step fails on the initial execution attempt. initial\_retry\_interval\_seconds is int, with default value of 1.

[ **@maximum\_retry\_interval\_seconds** = ] maximum\_retry\_interval\_seconds

The maximum delay between retry attempts. If the delay between retries would grow larger than this value, it is capped to this value instead. maximum\_retry\_interval\_seconds is int, with default value of 120.

[ **@retry\_interval\_backoff\_multiplier** = ] retry\_interval\_backoff\_multiplier

The multiplier to apply to the retry delay if multiple job step execution attempts fail. For example, if the first retry had a delay of 5 second and the backoff multiplier is 2.0, then the second retry will have a delay of 10 seconds and the third retry will have a delay of 20 seconds. retry\_interval\_backoff\_multiplier is real, with default value of 2.0.

[ **@retry\_attempts** = ] retry\_attempts

The number of times to retry execution if the initial attempt fails. For example, if the retry\_attempts value is 10, then there will be 1 initial attempt and 10 retry attempts, giving a total of 11 attempts. If the final retry attempt fails, then the job execution will terminate with a lifecycle of Failed. retry\_attempts is int, with default value of 10.

[ **@step\_timeout\_seconds** = ] step\_timeout\_seconds

The maximum amount of time allowed for the step to execute. If this time is exceeded, then the job execution will terminate with a lifecycle of TimedOut. step\_timeout\_seconds is int, with default value of 43,200 seconds (12 hours).

[ **@output\_type** = ] 'output\_type'

If not null, the type of destination that the command's first result set is written to. To reset the value of output\_type back to NULL, set this parameter's value to "" (empty string). output\_type is nvarchar(50), with a default of NULL.

If specified, the value must be SqlDatabase.

[ **@output\_credential\_name** = ] 'output\_credential\_name'

If not null, the name of the database scoped credential that is used to connect to the output destination database. Must be specified if output\_type equals SqlDatabase. To reset the value of output\_credential\_name back to NULL, set this parameter's value to "" (empty string). output\_credential\_name is nvarchar(128), with a default value of NULL.

[ **@output\_server\_name** = ] 'output\_server\_name'

If not null, the fully qualified DNS name of the server that contains the output destination database. Must be specified if output\_type equals SqlDatabase. To reset the value of output\_server\_name back to NULL, set this parameter's value to "" (empty string). output\_server\_name is nvarchar(256), with a default of NULL.

[ **@output\_database\_name** = ] 'output\_database\_name'

If not null, the name of the database that contains the output destination table. Must be specified if output\_type equals SqlDatabase. To reset the value of output\_database\_name back to NULL, set this parameter's value to "" (empty string). output\_database\_name is nvarchar(128), with a default of NULL.

[ **@output\_schema\_name** = ] 'output\_schema\_name'

If not null, the name of the SQL schema that contains the output destination table. If output\_type equals SqlDatabase, the default value is dbo. To reset the value of output\_schema\_name back to NULL, set this parameter's value to "" (empty string). output\_schema\_name is nvarchar(128).

[ **@output\_table\_name** = ] 'output\_table\_name'

If not null, the name of the table that the command's first result set will be written to. If the table doesn't already exist, it will be created based on the schema of the returning result-set. Must be specified if output\_type equals SqlDatabase. To reset the value of output\_table\_name back to NULL, set this parameter's value to "" (empty string). output\_table\_name is nvarchar(128), with a default value of NULL.

[ **@job\_version** = ] job\_version OUTPUT

Output parameter that will be assigned the new job version number. job\_version is int.

[ **@max\_parallelism** = ] max\_parallelism OUTPUT

The maximum level of parallelism per elastic pool. If set, then the job step will be restricted to only run on a maximum of that many databases per elastic pool. This applies to each elastic pool that is either directly included in the target group or is inside a server that is included in the target group. To reset the value of max\_parallelism back to null, set this parameter's value to -1. max\_parallelism is int.

#### Return Code Values

0 (success) or 1 (failure)

#### Remarks

Any in-progress executions of the job will not be affected. When sp\_update\_jobstep succeeds, the job's version number is incremented. The next time the job is executed, the new version will be used.

#### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users

## **sp\_delete\_jobstep**

Removes a job step from a job.

### **Syntax**

```
[jobs].sp_delete_jobstep [@job_name =] 'job_name'
[, [@step_id =] step_id]
[, [@step_name =] 'step_name']
[, [@job_version =] job_version OUTPUT]
```

### **Arguments**

**[ @job\_name = ] 'job\_name'**

The name of the job from which the step will be removed. job\_name is nvarchar(128), with no default.

**[ @step\_id = ] step\_id**

The identification number for the job step to be deleted. Either step\_id or step\_name must be specified. step\_id is an int.

**[ @step\_name = ] 'step\_name'**

The name of the step to be deleted. Either step\_id or step\_name must be specified. step\_name is nvarchar(128).

**[ @job\_version = ] job\_version OUTPUT**

Output parameter that will be assigned the new job version number. job\_version is int.

### **Return Code Values**

0 (success) or 1 (failure)

### **Remarks**

Any in-progress executions of the job will not be affected. When sp\_update\_jobstep succeeds, the job's version number is incremented. The next time the job is executed, the new version will be used.

The other job steps will be automatically renumbered to fill the gap left by the deleted job step.

### **Permissions**

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

## **sp\_start\_job**

Starts executing a job.

### **Syntax**

```
[jobs].sp_start_job [@job_name =] 'job_name'
[, [@job_execution_id =] job_execution_id OUTPUT]
```

### **Arguments**

**[ @job\_name = ] 'job\_name'**

The name of the job from which the step will be removed. job\_name is nvarchar(128), with no default.

**[ @job\_execution\_id = ] job\_execution\_id OUTPUT**

Output parameter that will be assigned the job execution's id. job\_version is uniqueidentifier.

### **Return Code Values**

0 (success) or 1 (failure)

#### Remarks

None.

#### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### **sp\_stop\_job**

Stops a job execution.

#### Syntax

```
[jobs].sp_stop_job [@job_execution_id =] 'job_execution_id '
```

#### Arguments

[ **@job\_execution\_id** = ] job\_execution\_id

The identification number of the job execution to stop. job\_execution\_id is uniqueidentifier, with default of NULL.

#### Return Code Values

0 (success) or 1 (failure)

#### Remarks

None.

#### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### **sp\_add\_target\_group**

Adds a target group.

#### Syntax

```
[jobs].sp_add_target_group [@target_group_name =] 'target_group_name'
[, [@target_group_id =] target_group_id OUTPUT]
```

#### Arguments

[ **@target\_group\_name** = ] 'target\_group\_name'

The name of the target group to create. target\_group\_name is nvarchar(128), with no default.

[ **@target\_group\_id** = ] target\_group\_id OUTPUT The target group identification number assigned to the job if created successfully. target\_group\_id is an output variable of type uniqueidentifier, with a default of NULL.

#### Return Code Values

0 (success) or 1 (failure)

## Remarks

Target groups provide an easy way to target a job at a collection of databases.

## Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

## sp\_delete\_target\_group

Deletes a target group.

### Syntax

```
[jobs].sp_delete_target_group [@target_group_name =] 'target_group_name'
```

### Arguments

[ **@target\_group\_name** = ] 'target\_group\_name'

The name of the target group to delete. target\_group\_name is nvarchar(128), with no default.

### Return Code Values

0 (success) or 1 (failure)

## Remarks

None.

## Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

## sp\_add\_target\_group\_member

Adds a database or group of databases to a target group.

### Syntax

```
[jobs].sp_add_target_group_member [@target_group_name =] 'target_group_name'
 [@membership_type =] 'membership_type']
 [, [@target_type =] 'target_type']
 [, [@refresh_credential_name =] 'refresh_credential_name']
 [, [@server_name =] 'server_name']
 [, [@database_name =] 'database_name']
 [, [@elastic_pool_name =] 'elastic_pool_name']
 [, [@shard_map_name =] 'shard_map_name']
 [, [@target_id =] 'target_id' OUTPUT]
```

### Arguments

[ **@target\_group\_name** = ] 'target\_group\_name'

The name of the target group to which the member will be added. target\_group\_name is nvarchar(128), with no default.

[ **@membership\_type** = ] 'membership\_type'

Specifies if the target group member will be included or excluded. target\_group\_name is nvarchar(128), with default of 'Include'. Valid values for target\_group\_name are 'Include' or 'Exclude'.

[ **@target\_type** = ] 'target\_type'

The type of target database or collection of databases including all databases in a server, all databases in an Elastic pool, all databases in a shard map, or an individual database. target\_type is nvarchar(128), with no default. Valid values for target\_type are 'SqlServer', 'SqlElasticPool', 'SqlDatabase', or 'SqlShardMap'.

[ **@refresh\_credential\_name** = ] 'refresh\_credential\_name'

The name of the SQL Database server. refresh\_credential\_name is nvarchar(128), with no default.

[ **@server\_name** = ] 'server\_name'

The name of the SQL Database server that should be added to the specified target group. server\_name should be specified when target\_type is 'SqlServer'. server\_name is nvarchar(128), with no default.

[ **@database\_name** = ] 'database\_name'

The name of the database that should be added to the specified target group. database\_name should be specified when target\_type is 'SqlDatabase'. database\_name is nvarchar(128), with no default.

[ **@elastic\_pool\_name** = ] 'elastic\_pool\_name'

The name of the Elastic pool that should be added to the specified target group. elastic\_pool\_name should be specified when target\_type is 'SqlElasticPool'. elastic\_pool\_name is nvarchar(128), with no default.

[ **@shard\_map\_name** = ] 'shard\_map\_name'

The name of the shard map pool that should be added to the specified target group. elastic\_pool\_name should be specified when target\_type is 'SqlSqlShardMap'. shard\_map\_name is nvarchar(128), with no default.

[ **@target\_id** = ] target\_group\_id OUTPUT

The target identification number assigned to the target group member if created added to the target group. target\_id is an output variable of type uniqueidentifier, with a default of NULL. Return Code Values 0 (success) or 1 (failure)

#### Remarks

A job executes on all single databases within a SQL Database server or in an elastic pool at time of execution, when a SQL Database server or Elastic pool is included in the target group.

#### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

#### Examples

The following example adds all the databases in the London and NewYork servers to the group Servers Maintaining Customer Information. You must connect to the jobs database specified when creating the job agent, in this case ElasticJobs.

```

--Connect to the jobs database specified when creating the job agent
USE ElasticJobs ;
GO

-- Add a target group containing server(s)
EXEC jobs.sp_add_target_group @target_group_name = N'Servers Maintaining Customer Information'
GO

-- Add a server target member
EXEC jobs.sp_add_target_group_member
@target_group_name = N'Servers Maintaining Customer Information',
@target_type = N'SqlServer',
@refresh credential_name=N'mymastercred', --credential required to refresh the databases in server
@server_name=N'London.database.windows.net' ;
GO

-- Add a server target member
EXEC jobs.sp_add_target_group_member
@target_group_name = N'Servers Maintaining Customer Information',
@target_type = N'SqlServer',
@refresh credential_name=N'mymastercred', --credential required to refresh the databases in server
@server_name=N'NewYork.database.windows.net' ;
GO

--View the recently added members to the target group
SELECT * FROM [jobs].target_group_members WHERE target_group_name= N'Servers Maintaining Customer
Information';
GO

```

## **sp\_delete\_target\_group\_member**

Removes a target group member from a target group.

### **Syntax**

```
[jobs].sp_delete_target_group_member [@target_group_name =] 'target_group_name'
[, [@target_id =] 'target_id']
```

Arguments [ @target\_group\_name = ] 'target\_group\_name'

The name of the target group from which to remove the target group member. target\_group\_name is nvarchar(128), with no default.

[ @target\_id = ] target\_id

The target identification number assigned to the target group member to be removed. target\_id is a uniqueidentifier, with a default of NULL.

### **Return Code Values**

0 (success) or 1 (failure)

### **Remarks**

Target groups provide an easy way to target a job at a collection of databases.

### **Permissions**

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### **Examples**

The following example removes the London server from the group Servers Maintaining Customer Information. You must connect to the jobs database specified when creating the job agent, in this case ElasticJobs.

```
--Connect to the jobs database specified when creating the job agent
USE ElasticJobs ;
GO

-- Retrieve the target_id for a target_group_members
declare @tid uniqueidentifier
SELECT @tid = target_id FROM [jobs].target_group_members WHERE target_group_name = 'Servers Maintaining
Customer Information' and server_name = 'London.database.windows.net'

-- Remove a target group member of type server
EXEC jobs.sp_delete_target_group_member
@target_group_name = N'Servers Maintaining Customer Information',
@target_id = @tid
GO
```

## sp\_purge\_jobhistory

Removes the history records for a job.

### Syntax

```
[jobs].sp_purge_jobhistory [@job_name =] 'job_name'
[, [@job_id =] job_id]
[, [@oldest_date =] oldest_date []]
```

### Arguments

[ **@job\_name** = ] 'job\_name'

The name of the job for which to delete the history records. job\_name is nvarchar(128), with a default of NULL. Either job\_id or job\_name must be specified, but both cannot be specified.

[ **@job\_id** = ] job\_id

The job identification number of the job for the records to be deleted. job\_id is uniqueidentifier, with a default of NULL. Either job\_id or job\_name must be specified, but both cannot be specified.

[ **@oldest\_date** = ] oldest\_date

The oldest record to retain in the history. oldest\_date is DATETIME2, with a default of NULL. When oldest\_date is specified, sp\_purge\_jobhistory only removes records that are older than the value specified.

### Return Code Values

0 (success) or 1 (failure) Remarks Target groups provide an easy way to target a job at a collection of databases.

### Permissions

By default, members of the sysadmin fixed server role can execute this stored procedure. They restrict a user to just be able to monitor jobs, you can grant the user to be part of the following database role in the job agent database specified when creating the job agent:

- jobs\_reader

For details about the permissions of these roles, see the Permission section in this document. Only members of sysadmin can use this stored procedure to edit the attributes of jobs that are owned by other users.

### Examples

The following example adds all the databases in the London and NewYork servers to the group Servers Maintaining Customer Information. You must connect to the jobs database specified when creating the job agent, in this case ElasticJobs.

```
--Connect to the jobs database specified when creating the job agent

EXEC sp_delete_target_group_member
 @target_group_name = N'Servers Maintaining Customer Information',
 @server_name = N'London.database.windows.net';
GO
```

## Job views

The following views are available in the [jobs database](#).

VIEW	DESCRIPTION
<a href="#">job_executions</a>	Shows job execution history.
<a href="#">jobs</a>	Shows all jobs.
<a href="#">job_versions</a>	Shows all job versions.
<a href="#">jobsteps</a>	Shows all steps in the current version of each job.
<a href="#">jobstep_versions</a>	Shows all steps in all versions of each job.
<a href="#">target_groups</a>	Shows all target groups.
<a href="#">target_group_members</a>	Shows all members of all target groups.

### **job\_executions view**

[jobs].[job\_executions]

Shows job execution history.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>job_execution_id</b>	uniqueidentifier	Unique ID of an instance of a job execution.
<b>job_name</b>	nvarchar(128)	Name of the job.
<b>job_id</b>	uniqueidentifier	Unique ID of the job.
<b>job_version</b>	int	Version of the job (automatically updated each time the job is modified).
<b>step_id</b>	int	Unique (for this job) identifier for the step. NULL indicates this is the parent job execution.
<b>is_active</b>	bit	Indicates whether information is active or inactive. 1 indicates active jobs, and 0 indicates inactive.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>lifecycle</b>	nvarchar(50)	Value indicating the status of the job: 'Created', 'In Progress', 'Failed', 'Succeeded', 'Skipped', 'SucceededWithSkipped'
<b>create_time</b>	datetime2(7)	Date and time the job was created.
<b>start_time</b>	datetime2(7)	Date and time the job started execution. NULL if the job has not yet been executed.
<b>end_time</b>	datetime2(7)	Date and time the job finished execution. NULL if the job has not yet been executed or has not yet completed execution.
<b>current_attempts</b>	int	Number of times the step was retried. Parent job will be 0, child job executions will be 1 or greater based on the execution policy.
<b>current_attempt_start_time</b>	datetime2(7)	Date and time the job started execution. NULL indicates this is the parent job execution.
<b>last_message</b>	nvarchar(max)	Job or step history message.
<b>target_type</b>	nvarchar(128)	Type of target database or collection of databases including all databases in a server, all databases in an Elastic pool or a database. Valid values for target_type are 'SqlServer', 'SqlElasticPool' or 'SqlDatabase'. NULL indicates this is the parent job execution.
<b>target_id</b>	uniqueidentifier	Unique ID of the target group member. NULL indicates this is the parent job execution.
<b>target_group_name</b>	nvarchar(128)	Name of the target group. NULL indicates this is the parent job execution.
<b>target_server_name</b>	nvarchar(256)	Name of the SQL Database server contained in the target group. Specified only if target_type is 'SqlServer'. NULL indicates this is the parent job execution.
<b>target_database_name</b>	nvarchar(128)	Name of the database contained in the target group. Specified only when target_type is 'SqlDatabase'. NULL indicates this is the parent job execution.

## **jobs view**

[jobs].[jobs]

Shows all jobs.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>job_name</b>	nvarchar(128)	Name of the job.
<b>job_id</b>	uniqueidentifier	Unique ID of the job.
<b>job_version</b>	int	Version of the job (automatically updated each time the job is modified).
<b>description</b>	nvarchar(512)	Description for the job. enabled bit Indicates whether the job is enabled or disabled. 1 indicates enabled jobs, and 0 indicates disabled jobs.
<b>schedule_interval_type</b>	nvarchar(50)	Value indicating when the job is to be executed:'Once', 'Minutes', 'Hours', 'Days', 'Weeks', 'Months'
<b>schedule_interval_count</b>	int	Number of schedule_interval_type periods to occur between each execution of the job.
<b>schedule_start_time</b>	datetime2(7)	Date and time the job was last started execution.
<b>schedule_end_time</b>	datetime2(7)	Date and time the job was last completed execution.

## **job\_versions view**

[jobs].[job\_versions]

Shows all job versions.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>job_name</b>	nvarchar(128)	Name of the job.
<b>job_id</b>	uniqueidentifier	Unique ID of the job.
<b>job_version</b>	int	Version of the job (automatically updated each time the job is modified).

## **jobsteps view**

[jobs].[jobsteps]

Shows all steps in the current version of each job.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>job_name</b>	nvarchar(128)	Name of the job.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>job_id</b>	uniqueidentifier	Unique ID of the job.
<b>job_version</b>	int	Version of the job (automatically updated each time the job is modified).
<b>step_id</b>	int	Unique (for this job) identifier for the step.
<b>step_name</b>	nvarchar(128)	Unique (for this job) name for the step.
<b>command_type</b>	nvarchar(50)	Type of command to execute in the job step. For v1, value must equal to and defaults to 'TSql'.
<b>command_source</b>	nvarchar(50)	Location of the command. For v1, 'Inline' is the default and only accepted value.
<b>command</b>	nvarchar(max)	The commands to be executed by Elastic jobs through command_type.
<b>credential_name</b>	nvarchar(128)	Name of the database scoped credential used to execution the job.
<b>target_group_name</b>	nvarchar(128)	Name of the target group.
<b>target_group_id</b>	uniqueidentifier	Unique ID of the target group.
<b>initial_retry_interval_seconds</b>	int	The delay before the first retry attempt. Default value is 1.
<b>maximum_retry_interval_seconds</b>	int	The maximum delay between retry attempts. If the delay between retries would grow larger than this value, it is capped to this value instead. Default value is 120.
<b>retry_interval_backoff_multiplier</b>	real	The multiplier to apply to the retry delay if multiple job step execution attempts fail. Default value is 2.0.
<b>retry_attempts</b>	int	The number of retry attempts to use if this step fails. Default of 10, which indicates no retry attempts.
<b>step_timeout_seconds</b>	int	The amount of time in minutes between retry attempts. The default is 0, which indicates a 0-minute interval.
<b>output_type</b>	nvarchar(11)	Location of the command. In the current preview, 'Inline' is the default and only accepted value.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>output_credential_name</b>	nvarchar(128)	Name of the credentials to be used to connect to the destination server to store the results set.
<b>output_subscription_id</b>	uniqueidentifier	Unique ID of the subscription of the destination server\database for the results set from the query execution.
<b>output_resource_group_name</b>	nvarchar(128)	Resource group name where the destination server resides.
<b>output_server_name</b>	nvarchar(256)	Name of the destination server for the results set.
<b>output_database_name</b>	nvarchar(128)	Name of the destination database for the results set.
<b>output_schema_name</b>	nvarchar(max)	Name of the destination schema. Defaults to dbo, if not specified.
<b>output_table_name</b>	nvarchar(max)	Name of the table to store the results set from the query results. Table will be created automatically based on the schema of the results set if it doesn't already exist. Schema must match the schema of the results set.
<b>max_parallelism</b>	int	The maximum number of databases per elastic pool that the job step will be run on at a time. The default is NULL, meaning no limit.

#### **jobstep\_versions view**

[jobs].[jobstep\_versions]

Shows all steps in all versions of each job. The schema is identical to [jobsteps](#).

#### **target\_groups view**

[jobs].[target\_groups]

Lists all target groups.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>target_group_name</b>	nvarchar(128)	The name of the target group, a collection of databases.
<b>target_group_id</b>	uniqueidentifier	Unique ID of the target group.

#### **target\_group\_members view**

[jobs].[target\_group\_members]

Shows all members of all target groups.

COLUMN NAME	DATA TYPE	DESCRIPTION
<b>target_group_name</b>	nvarchar(128)	The name of the target group, a collection of databases.
<b>target_group_id</b>	uniqueidentifier	Unique ID of the target group.
<b>membership_type</b>	int	Specifies if the target group member is included or excluded in the target group. Valid values for target_group_name are 'Include' or 'Exclude'.
<b>target_type</b>	nvarchar(128)	Type of target database or collection of databases including all databases in a server, all databases in an Elastic pool or a database. Valid values for target_type are 'SqlServer', 'SqlElasticPool', 'SqlDatabase', or 'SqlShardMap'.
<b>target_id</b>	uniqueidentifier	Unique ID of the target group member.
<b>refresh_credential_name</b>	nvarchar(128)	Name of the database scoped credential used to connect to the target group member.
<b>subscription_id</b>	uniqueidentifier	Unique ID of the subscription.
<b>resource_group_name</b>	nvarchar(128)	Name of the resource group in which the target group member resides.
<b>server_name</b>	nvarchar(128)	Name of the SQL Database server contained in the target group. Specified only if target_type is 'SqlServer'.
<b>database_name</b>	nvarchar(128)	Name of the database contained in the target group. Specified only when target_type is 'SqlDatabase'.
<b>elastic_pool_name</b>	nvarchar(128)	Name of the Elastic pool contained in the target group. Specified only when target_type is 'SqlElasticPool'.
<b>shard_map_name</b>	nvarchar(128)	Name of the shard map contained in the target group. Specified only when target_type is 'SqlShardMap'.

## Resources

-  [Transact-SQL Syntax Conventions](#)

## Next steps

- [Create and manage Elastic Jobs using PowerShell](#)
- [Authorization and Permissions SQL Server](#)

# Migrate to the new Elastic Database jobs

11/7/2019 • 12 minutes to read • [Edit Online](#)

An upgraded version of [Elastic Database Jobs](#) is available.

If you have an existing customer hosted version of Elastic Database Jobs, migration cmdlets and scripts are provided for easily migrating to the latest version.

## Prerequisites

The upgraded version of Elastic Database jobs has a new set of PowerShell cmdlets for use during migration. These new cmdlets transfer all of your existing job credentials, targets (including databases, servers, custom collections), job triggers, job schedules, job contents, and jobs over to a new Elastic Job agent.

### Install the latest Elastic Jobs cmdlets

If you don't already have an Azure subscription, [create a free account](#) before you begin.

Install the **Az.Sql** 1.1.1-preview module to get the latest Elastic Job cmdlets. Run the following commands in PowerShell with administrative access.

```
Installs the latest PackageManagement powershell package which PowerShellGet v1.6.5 is dependent on
Find-Package PackageManagement -RequiredVersion 1.1.7.2 | Install-Package -Force

Installs the latest PowerShellGet module which adds the -AllowPrerelease flag to Install-Module
Find-Package PowerShellGet -RequiredVersion 1.6.5 | Install-Package -Force

Restart your powershell session with administrative access

Places Az.Sql preview cmdlets side by side with existing Az.Sql version
Install-Module -Name Az.Sql -RequiredVersion 1.1.1-preview -AllowPrerelease

Import the Az.Sql module
Import-Module Az.Sql -RequiredVersion 1.1.1

Confirm if module successfully imported - if the imported version is 1.1.1, then continue
Get-Module Az.Sql
```

### Create a new Elastic Job agent

After installing the new cmdlets, create a new Elastic Job agent.

```
Register your subscription for the for the Elastic Jobs public preview feature
Register-AzProviderFeature -FeatureName sqldb-JobAccounts -ProviderNamespace Microsoft.Sql

Get an existing database to use as the job database - or create a new one if necessary
$db = Get-AzSqlDatabase -ResourceGroupName <resourceGroupName> -ServerName <serverName> -DatabaseName
<databaseName>
Create a new elastic job agent
$agent = $db | New-AzSqlElasticJobAgent -Name <agentName>
```

### Install the old Elastic Database Jobs cmdlets

Migration needs to use some of the *old* elastic job cmdlets, so run the following commands if you don't already have them installed.

```

Install the old elastic job cmdlets if necessary and initialize the old jobs cmdlets
.\nuget install Microsoft.Azure.SqlDatabase.Jobs -prerelease

Install the old jobs cmdlets
cd Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*\tools
Unblock-File .\InstallElasticDatabaseJobsCmdlets.ps1
.\InstallElasticDatabaseJobsCmdlets.ps1

Choose the subscription where your existing jobs are
Select-AzSubscription -SubscriptionId <subscriptionId>
Use-AzureSqlJobConnection -CurrentAzureSubscription -Credential (Get-Credential)

```

## Migration

Now that both the old and new Elastic Jobs cmdlets are initialized, migrate your job credentials, targets, and jobs to the new *job database*.

### Setup

```

$ErrorActionPreference = "Stop";

Helper function to show starting write output
function Log-StartOutput ($output) {
 Write-Output ("`r----- " + $output + " -----")
}

Helper function to show starting write output
function Log-ChildOutput ($output) {
 Write-Output (" - " + $output)
}

```

### Migrate credentials

```

function Migrate-Credentials ($agent) {
 Log-StartOutput "Migrating credentials"

 $oldCreds = Get-AzureSqlJobCredential
 $oldCreds | % {
 $oldCredName = $_.CredentialName
 $oldUserName = $_.UserName
 Write-Output ("Credential " + $oldCredName)
 $oldCredential = Get-Credential -UserName $oldUserName `

 -Message ("Please enter in the password that was used for your credential " +
$oldCredName)
 try
 {
 $cred = New-AzSqlElasticJobCredential -ParentObject $agent -Name $oldCredName -Credential
$oldCredential
 }
 catch [System.Management.Automation.PSArgumentException]
 {
 $cred = Get-AzSqlElasticJobCredential -ParentObject $agent -Name $oldCredName
 $cred = Set-AzSqlElasticJobCredential -InputObject $cred -Credential $oldCredential
 }

 Log-ChildOutput ("Added user " + $oldUserName)
 }
}

```

To migrate your credentials, execute the following command by passing in the `$agent` PowerShell object from earlier.

```
Migrate-Credentials $agent
```

## Sample output

```
You should see similar output after executing the above
----- Migrating credentials -----
Credential cred1
- Added user user1
Credential cred2
- Added user user2
Credential cred3
- Added user user3
```

## Migrate targets

```
function Migrate-TargetGroups ($agent) {
 Log-StartOutput "Migrating target groups"

 # Setup hash of target groups
 $targetGroups = [ordered]@{}

 # Fetch root job targets from old service
 $rootTargets = Get-AzureSqlJobTarget

 # Return if no root targets are found
 if ($rootTargets.Count -eq 0)
 {
 Write-Output "No targets found - no need for migration"
 return
 }

 # Create list of target groups to create
 # We format the target group name as such:
 # - If root target is server type, then target group name is "(serverName)"
 # - If root target is database type, then target group name is "(serverName,databaseName)"
 # - If root target is shard map type, then target group name is "(serverName,databaseName,shardMapName)"
 # - If root target is custom collection, then target group name is "customCollectionName"
 $rootTargets | % {
 $tgName = Format-OldTargetName -target $_
 $childTargets = Get-ChildTargets -target $_
 $targetGroups.Add($tgName, $childTargets)
 }

 # Flatten list
 for ($i=$targetGroups.Count - 1; $i -ge 0; $i--)
 {
 # Fetch target group's initial list of targets unexpanded
 $targets = $targetGroups[$i]

 # Expand custom collection targets
 $j = 0;
 while ($j -lt $targets.Count)
 {
 $target = $targets[$j]
 if ($target.TargetType -eq "CustomCollection")
 {
 $targets = [System.Collections.ArrayList] $targets
 $targets.Remove($target) # Remove this target from the list

 $expandedTargets = $targetGroups[$target.TargetDescription.CustomCollectionName]

 foreach ($expandedTarget in $expandedTargets)
 {
 $targets.Add($expandedTarget) | Out-Null
 }
 }
 }
 }
}
```

```

 }

 # Set updated list of targets for tg
 $targetGroups[$i] = $targets
 # Note we don't increment here in case we need to expand further
 }
 else
 {
 # Skip if no custom collection target needs to be expanded
 $j++
 }
}

Add targets to target group
foreach ($targetGroup in $targetGroups.Keys)
{
 $tg = Setup-TargetGroup -tgName $targetGroup -agent $agent
 $targets = $targetGroups[$targetGroup]
 Migrate-Targets -targets $targets -tg $tg
 $targetsAdded = (Get-AzSqlElasticJobTargetGroup -ParentObject $agent -Name
$tg.TargetGroupName).Targets
 foreach ($targetAdded in $targetsAdded)
 {
 Log-ChildOutput ("Added target " + (Format-NewTargetName $targetAdded))
 }
}
}

Target group helpers
Migrate shard map target from old jobs to new job's target group
function Migrate-Targets ($targets, $tg) {
 Write-Output ("Target group " + $tg.TargetGroupName)
 foreach ($target in $targets) {
 if ($target.TargetType -eq "Server") {
 Add-ServerTarget -target $target -tg $tg
 }
 elseif ($target.TargetType -eq "Database") {
 Add-DatabaseTarget -target $target -tg $tg
 }
 elseif ($target.TargetType -eq "ShardMap") {
 Add-ShardMapTarget -target $target -tg $tg
 }
 }
}

Migrate server target from old jobs to new job's target group
function Add-ServerTarget ($target, $tg) {
 $jobTarget = Get-AzureSqlJobTarget -TargetId $target.TargetId
 $serverName = $jobTarget.ServerName
 $credName = $jobTarget.MasterDatabaseCredentialName
 $t = Add-AzSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -RefreshCredentialName $credName
}

Migrate database target from old jobs to new job's target group
function Add-DatabaseTarget ($target, $tg) {
 $jobTarget = Get-AzureSqlJobTarget -TargetId $target.TargetId
 $serverName = $jobTarget.ServerName
 $databaseName = $jobTarget.DatabaseName
 $exclude = $target.Membership

 if ($exclude -eq "Exclude") {
 $t = Add-AzSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -DatabaseName $databaseName -
Exclude
 }
 else {
 $t = Add-AzSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -DatabaseName $databaseName
 }
}

```

```

Migrate shard map target from old jobs to new job's target group
function Add-ShardMapTarget ($target, $tg) {
 $jobTarget = Get-AzureSqlJobTarget -TargetId $target.TargetId
 $smName = $jobTarget.ShardMapName
 $serverName = $jobTarget.ShardMapManagerServerName
 $databaseName = $jobTarget.ShardMapManagerDatabaseName
 $credName = $jobTarget.ShardMapManagerCredentialName
 $exclude = $target.Membership

 if ($exclude -eq "Exclude") {
 $t = Add-AzSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -ShardMapName $smName -
 DatabaseName $databasename -RefreshCredentialName $credName -Exclude
 }
 else {
 $t = Add-AzSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -ShardMapName $smName -
 DatabaseName $databasename -RefreshCredentialName $credName
 }
}

Helper to format target old target names
function Format-OldTargetName ($target) {
 if ($target.TargetType -eq "Server") {
 $tgName = "(" + $target.ServerName + ")"
 }
 elseif ($target.TargetType -eq "Database") {
 $tgName = "(" + $target.ServerName + "," + $target.DatabaseName + ")"
 }
 elseif ($target.TargetType -eq "ShardMap") {
 $tgName = "(" + $target.ShardMapManagerServerName + "," +
 $target.ShardMapManagerDatabaseName + "," +
 $target.ShardMapName + ")"
 }
 elseif ($target.TargetType -eq "CustomCollection") {
 $tgName = $target.CustomCollectionName
 }
}

return $tgName
}

Helper to format new target names
function Format-NewTargetName ($target) {
 if ($target.TargetType -eq "SqlServer") {
 $tgName = "(" + $target.TargetServerName + ")"
 }
 elseif ($target.TargetType -eq "SqlDatabase") {
 $tgName = "(" + $target.TargetServerName + "," + $target.TargetDatabaseName + ")"
 }
 elseif ($target.TargetType -eq "SqlShardMap") {
 $tgName = "(" + $target.TargetServerName + "," +
 $target.TargetDatabaseName + "," +
 $target.TargetShardMapName + ")"
 }
 elseif ($target.TargetType -eq "SqlElasticPool") {
 $tgName = "(" + $target.TargetServerName + "," +
 $target.TargetDatabaseName + "," +
 $target.TargetElasticPoolName + ")"
 }

 return $tgName
}

Get child targets
function Get-ChildTargets($target) {
 if ($target.TargetType -eq "CustomCollection") {
 $children = Get-AzureSqlJobChildTarget -TargetId $target.TargetId
 if ($children.Count -eq 1)
 {
 $arr = New-Object System.Collections.ArrayList($null)

```

```

 $arr.Add($children)
 $children = $arr
 }
 return $children
}
else {
 return $target
}
}

Migrates target groups
function Setup-TargetGroup ($tgName, $agent) {
 try {
 $tg = New-AzSqlElasticJobTargetGroup -ParentObject $agent -Name $tgName
 return $tg
 }
 catch [System.Management.Automation.PSArgumentException] {
 $tg = Get-AzSqlElasticJobTargetGroup -ParentObject $agent -Name $tgName
 return $tg
 }
}

```

To migrate your targets (servers, databases, and custom collections) to your new job database, execute the **Migrate-TargetGroups** cmdlet to perform the following:

- Root level targets that are servers and databases will be migrated to a new target group named "(<serverName>, <databaseName>)" containing only the root level target.
- A custom collection will migrate to a new target group containing all child targets.

```
Migrate-TargetGroups $agent
```

Sample output:

```

----- Migrating target groups -----
Target group cc1
- Added target (s1)
- Added target (s1,db1)
Target group cc2
- Added target (s1,db1)
Target group cc3
- Added target (s1)
- Added target (s1,db1)
Target group (s1,db1)
- Added target (s1,db1)
Target group (s1,db2)
- Added target (s1,db2)
Target group (s1)
- Added target (s1)
Target group (s1,db1,sm1)
- Added target (s1,db1,sm1)

```

## Migrate jobs

```

function Migrate-Jobs ($agent)
{
 Log-StartOutput "Migrating jobs and job steps"

 $oldJobs = Get-AzureSqlJob
 $newJobs = [System.Collections.ArrayList] @()

 foreach ($oldJob in $oldJobs)
 {

```

```

ignore system jobs
if ($oldJob.ContentName -eq $null)
{
 continue
}

Schedule
$oldJobTriggers = Get-AzureSqlJobTrigger -JobName $oldJob.JobName

if ($oldJobTriggers.Count -ge 1)
{
 foreach ($trigger in $oldJobTriggers)
 {

 $schedule = Get-AzureSqlJobSchedule -ScheduleName $trigger.ScheduleName
 $newJob = [PSCustomObject] @@
 JobName = ($trigger.JobName + " (" + $trigger.ScheduleName + ")");
 Description = $oldJob.ContentName
 Schedule = $schedule
 TargetGroupName = (Format-OldTargetName(Get-AzureSqlJobTarget -TargetId $oldJob.TargetId))
 CredentialName = $oldJob.CredentialName
 Output = $oldJob.ResultSetDestination
 }
 $newJobs.Add($newJob) | Out-Null
}
else
{
 $newJob = [PSCustomObject] @@
 JobName = $oldJob.JobName
 Description = $oldJob.ContentName
 Schedule = $null
 TargetGroupName = (Format-OldTargetName(Get-AzureSqlJobTarget -TargetId $oldJob.TargetId))
 CredentialName = $oldJob.CredentialName
 Output = $oldJob.ResultSetDestination
 }
 $newJobs.Add($newJob) | Out-Null
}
}

At this point, we should have an organized list of jobs to create
foreach ($newJob in $newJobs)
{
 Write-Output ("Job " + $newJob.JobName)
 $job = Setup-Job $newJob $agent
 If ($job.Interval -ne $null)
 {
 Log-ChildOutput ("Schedule with start time " + $job.StartTime + " and end time at " +
$job.EndTime)
 Log-ChildOutput ("Repeats every " + $job.Interval)
 }
 else {
 Log-ChildOutput ("Repeats once")
 }

 Setup-JobStep $newJob $job
}
}

Migrates jobs
function Setup-Job ($job, $agent) {
 $jobName = $newJob.JobName
 $jobDescription = $newJob.Description

 # Create or update a job has a recurring schedule
 if ($newJob.Schedule -ne $null) {
 $schedule = $newJob.Schedule
 $startTime = $schedule.StartTime.UtcTime
 $endTime = $schedule.EndTime.UtcTime
 }
}

```

```

$intervalType = $schedule.Interval.IntervalType.ToString()
$intervalType = $intervalType.Substring(0, $intervalType.Length - 1) # Remove the last letter (s)
$intervalCount = $schedule.Interval.Count

try {
 $job = New-AzSqlElasticJob -ParentObject $agent -Name $jobName `
 -Description $jobDescription -IntervalType $intervalType -IntervalCount $intervalCount `
 -StartTime $startTime -EndTime $endTime
 return $job
}
catch [System.Management.Automation.PSArgumentException] {
 $job = Get-AzSqlElasticJob -ParentObject $agent -Name $jobName
 $job = $job | Set-AzSqlElasticJob -Description $jobDescription -IntervalType $intervalType -
 IntervalCount $intervalCount `
 -StartTime $startTime -EndTime $endTime
 return $job
}
}

Create or update a job that runs once
else {
 try {
 $job = New-AzSqlElasticJob -ParentObject $agent -Name $jobName `
 -Description $jobDescription -RunOnce
 return $job
 }
 catch [System.Management.Automation.PSArgumentException] {
 $job = Get-AzSqlElasticJob -ParentObject $agent -Name $jobName
 $job = $job | Set-AzSqlElasticJob -Description $jobDescription -RunOnce
 return $job
 }
}
}

Migrates job steps
function Setup-JobStep ($newJob, $job) {
 $defaultJobStepName = 'JobStep'
 $contentName = $newJob.Description
 $commandText = (Get-AzureSqlJobContentDefinition -ContentName $contentName).CommandText
 $targetGroupName = $newJob.TargetGroupName
 $credentialName = $newJob.CredentialName

 $output = $newJob.Output

 if ($output -ne $null) {
 $outputServerName = $output.TargetDescription.ServerName
 $outputDatabaseName = $output.TargetDescription.DatabaseName
 $outputCredentialName = $output.CredentialName
 $outputSchemaName = $output.SchemaName
 $outputTableName = $output.TableName
 $outputDatabase = Get-AzSqlDatabase -ResourceGroupName $job.ResourceGroupName -ServerName
 $outputServerName -Databasename $outputDatabaseName

 try {
 $jobStep = $job | Add-AzSqlElasticJobStep -Name $defaultJobStepName `
 -TargetGroupName $targetGroupName -CredentialName $credentialName -CommandText $commandText `
 -OutputDatabaseObject $outputDatabase `
 -OutputSchemaName $outputSchemaName -OutputTableName $outputTableName `
 -OutputCredentialName $outputCredentialName
 }
 catch [System.Management.Automation.PSArgumentException] {
 $jobStep = $job | Get-AzSqlElasticJobStep -Name $defaultJobStepName
 $jobStep = $jobStep | Set-AzSqlElasticJobStep -TargetGroupName $targetGroupName `
 -CredentialName $credentialName -CommandText $commandText `
 -OutputDatabaseObject $outputDatabase `
 -OutputSchemaName $outputSchemaName -OutputTableName $outputTableName `
 -OutputCredentialName $outputCredentialName
 }
 }
 else {
 try {

```

```

$jobStep = $job | Add-AzSqlElasticJobStep -Name $defaultJobStepName -TargetGroupName $targetGroupName -
CredentialName $credentialName -CommandText $commandText
}
catch [System.Management.Automation.PSArgumentException] {
 $jobStep = $job | Get-AzSqlElasticJobStep -Name $defaultJobStepName
 $jobStep = $jobStep | Set-AzSqlElasticJobStep -TargetGroupName $targetGroupName -CredentialName
$credentialName -CommandText $commandText
}
}
Log-ChildOutput ("Added step " + $jobStep.StepName + " using target group " + $jobStep.TargetGroupName + " "
using credential " + $jobStep.CredentialName)
Log-ChildOutput("Command text script taken from content name " + $contentName)

if ($jobStep.Output -ne $null) {
 Log-ChildOutput ("With output target as (" + $jobStep.Output.ServerName + "," +
$jobStep.Output.DatabaseName + "," + $jobStep.Output.SchemaName + "," + $jobStep.Output.TableName + ")")
}
}

```

To migrate your jobs, job content, job triggers, and job schedules over to your new Elastic Job agent's database, execute the **Migrate-Jobs** cmdlet passing in your agent.

- Jobs with multiple triggers with different schedules are separated into multiple jobs with naming scheme: "<jobName> (<scheduleName>)".
- Job contents are migrated to a job by adding a default job step named JobStep with associated command text.
- Jobs are disabled by default so that you can validate them before enabling them.

```
Migrate-Jobs $agent
```

Sample output:

```

----- Migrating jobs and job steps -----
Job job1
- Repeats once
- Added step JobStep using target group cc2 using credential cred1
- Command text script taken from content name SampleContent
Job job2
- Repeats once
- Added step JobStep using target group (s1,db1) using credential cred1
- Command text script taken from content name SampleContent
- With output target as (s1,db1,dbo,sampleTable)
Job job3 (repeat every 10 min)
- Schedule with start time 05/16/2018 22:05:28 and end time at 12/31/9999 11:59:59
- Repeats every PT10M
- Added step JobStep using target group cc1 using credential cred1
- Command text script taken from content name SampleContent
Job job3 (repeat every 5 min)
- Schedule with start time 05/16/2018 22:05:31 and end time at 12/31/9999 11:59:59
- Repeats every PT5M
- Added step JobStep using target group cc1 using credential cred1
- Command text script taken from content name SampleContent
Job job4
- Repeats once
- Added step JobStep using target group (s1,db1) using credential cred1
- Command text script taken from content name SampleContent

```

## Migration Complete

The *job database* should now have all of the job credentials, targets, job triggers, job schedules, job contents, and jobs migrated over.

To confirm that everything migrated correctly, use the following scripts:

```
$creds = $agent | Get-AzSqlElasticJobCredential
$targetGroups = $agent | Get-AzSqlElasticJobTargetGroup
$jobs = $agent | Get-AzSqlElasticJob
$steps = $jobs | Get-AzSqlElasticJobStep
```

To test that jobs are executing correctly, start them:

```
$jobs | Start-AzSqlElasticJob
```

For any jobs that were running on a schedule, remember to enable them so that they can run in the background:

```
$jobs | Set-AzSqlElasticJob -Enable
```

## Next steps

- [Create and manage Elastic Jobs using PowerShell](#)
- [Create and manage Elastic Jobs using Transact-SQL \(T-SQL\)](#)

# Azure CLI samples for Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

Azure SQL Database can be configured using [Azure CLI](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the CLI locally, this topic requires that you are running the Azure CLI version 2.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).

- [Single database & Elastic pools](#)
- [Managed Instance](#)

The following table includes links to Azure CLI script examples for Azure SQL Database.

<a href="#">Create a single database and an elastic pool</a>	
<a href="#">Create a single database and configure a firewall rule</a>	This CLI script example creates a single Azure SQL database and configures a server-level firewall rule.

Create elastic pools and move pooled databases	This CLI script example creates SQL elastic pools, and moves pooled Azure SQL databases, and changes compute sizes.
<b>Scale a single database and an elastic pool</b>	
Scale a single database	This CLI script example scales a single Azure SQL database to a different compute size after querying the size information for the database.
Scale an elastic pool	This CLI script example scales a SQL elastic pool to a different compute size.
<b>Failover groups</b>	
Add single database to failover group	This CLI script creates a database, and a failover group, adds the database to the failover group and tests failover to the secondary server.

Learn more about the [Single Database Azure CLI API](#).

# Azure PowerShell samples for Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

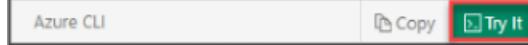
Azure SQL Database enables you to configure your databases, instances, and pools using Azure PowerShell.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this tutorial requires AZ PowerShell 1.4.0 or later. If you need to upgrade, see [Install Azure PowerShell module](#). If you are running PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

- [Single Database and Elastic pools](#)
- [Managed Instance](#)

The following table includes links to sample Azure PowerShell scripts for Azure SQL Database.

<b>Create and configure single databases, and elastic pools</b>	
<a href="#">Create a single database and configure a database server firewall rule</a>	This PowerShell script creates a single Azure SQL database and configures a server-level firewall rule.

<a href="#">Create elastic pools and move pooled databases</a>	This PowerShell script creates Azure SQL Database elastic pools, and moves pooled databases, and changes compute sizes.
<b>Configure geo-replication and failover</b>	
<a href="#">Configure and failover a single database using active geo-replication</a>	This PowerShell script configures active geo-replication for a single Azure SQL database and fails it over to the secondary replica.
<a href="#">Configure and failover a pooled database using active geo-replication</a>	This PowerShell script configures active geo-replication for an Azure SQL database in a SQL elastic pool, and fails it over to the secondary replica.
<b>Configure a failover group</b>	
<a href="#">Configure a failover group for a single database</a>	This PowerShell script creates a database, and a failover group, adds the database to the failover group and tests failover to the secondary server.
<a href="#">Configure a failover group for an elastic pool</a>	This PowerShell script creates a database, adds it to an elastic pool, adds the elastic pool to the failover group and tests failover to the secondary server.
<b>Scale a single database and an elastic pool</b>	
<a href="#">Scale a single database</a>	This PowerShell script monitors the performance metrics of an Azure SQL database, scales it to a higher compute size and creates an alert rule on one of the performance metrics.
<a href="#">Scale an elastic pool</a>	This PowerShell script monitors the performance metrics of an Azure SQL Database elastic pool, scales it to a higher compute size, and creates an alert rule on one of the performance metrics.
<b>Auditing and threat detection</b>	
<a href="#">Configure auditing and threat-detection</a>	This PowerShell script configures auditing and threat detection policies for an Azure SQL database.
<b>Restore, copy, and import a database</b>	
<a href="#">Restore a database</a>	This PowerShell script restores an Azure SQL database from a geo-redundant backup and restores a deleted Azure SQL database to the latest backup.
<a href="#">Copy a database to new server</a>	This PowerShell script creates a copy of an existing Azure SQL database in a new Azure SQL server.
<a href="#">Import a database from a bacpac file</a>	This PowerShell script imports a database to an Azure SQL server from a bacpac file.
<b>Sync data between databases</b>	

Sync data between SQL databases	This PowerShell script configures Data Sync to sync between multiple Azure SQL databases.
Sync data between SQL Database and SQL Server on-premises	This PowerShell script configures Data Sync to sync between an Azure SQL database and a SQL Server on-premises database.
Update the SQL Data Sync sync schema	This PowerShell script adds or removes items from the Data Sync sync schema.

Learn more about the [Single Database Azure PowerShell API](#).

## Additional resources

The examples listed on this page use the [Azure SQL Database cmdlets](#) for creating and managing Azure SQL resources. Additional cmdlets for running queries, and performing many database tasks are located in the `sqlserver` module. For more information, see [SQL Server PowerShell](#).

# Azure Resource Manager templates for Azure SQL Database

11/7/2019 • 3 minutes to read • [Edit Online](#)

Azure Resource Manager templates enable you to define your infrastructure as code and deploy your solutions to Azure cloud.

- [Single database & elastic pool](#)
- [Managed Instance](#)

The following table includes links to Azure Resource Manager templates for Azure SQL Database.

<a href="#">Single database</a>	This Azure Resource Manager template creates a single Azure SQL Database with logical server and configures firewall rules.
<a href="#">Logical server</a>	This Azure Resource Manager template creates a logical server for Azure SQL Database.
<a href="#">Elastic pool</a>	This template allows you to deploy a new Elastic pool with its new associated SQL Server and new SQL Databases to assign to it.
<a href="#">Failover groups</a>	This template creates two Azure SQL logical servers, a SQL database, and a failover group.
<a href="#">Threat Detection</a>	This template allows you to deploy an Azure SQL logical server and a set of Azure SQL Databases with Threat Detection enabled, with an email address for alerts for each database. Threat Detection is part of the SQL Advanced Threat Protection (ATP) offering and provides a layer of security that responds to potential threats over SQL servers and databases.
<a href="#">Auditing to Azure Blob Storage</a>	This template allows you to deploy an Azure SQL logical server with Auditing enabled to write audit logs to a blob storage. Auditing for Azure SQL Database tracks database events and writes them to an audit log that can be placed in your Azure storage account, OMS workspace, or Event Hubs.
<a href="#">Auditing to Azure Event Hub</a>	This template allows you to deploy an Azure SQL server with Auditing enabled to write audit logs to an existing Event Hub. In order to send audit events to Event Hub, set auditing settings with <code>Enabled State</code> and set <code>IsAzureMonitorTargetEnabled</code> as <code>true</code> . Also, configure Diagnostic Settings with <code>SQLSecurityAuditEvents</code> diagnostic logs category on the <code>master</code> database (for serve level auditing). Auditing for Azure SQL Database and SQL Data Warehouse tracks database events and writes them to an audit log that can be placed in your Azure storage account, OMS workspace, or Event Hubs.
<a href="#">Azure Web App with SQL Database</a>	This sample creates a free Azure Web App and SQL Database at the "Basic" service level.

Azure Web App and Redis Cache with SQL Database	This template creates a Web App, Redis Cache, and SQL Database in the same resource group, and creates two connection strings in the Web App for the SQL Database and Redis Cache.
Import data from blob storage using ADF V2	This Azure Resource Manager template creates Azure Data Factory V2 that copies data from Azure Blob Storage to SQL Database.
HDInsight cluster with a SQL Database	This template allows you to create a HDInsight cluster, a SQL Database server, a SQL Database, and two tables. This template is used by the <a href="#">Use Sqoop with Hadoop in HDInsight article</a>
Azure Logic App that runs a SQL Stored Procedure on a schedule	This template allows you to create a Logic App that will run a SQL stored procedure on schedule. Any arguments for the procedure can be put into the body section of the template.

# What is Azure SQL Database managed instance?

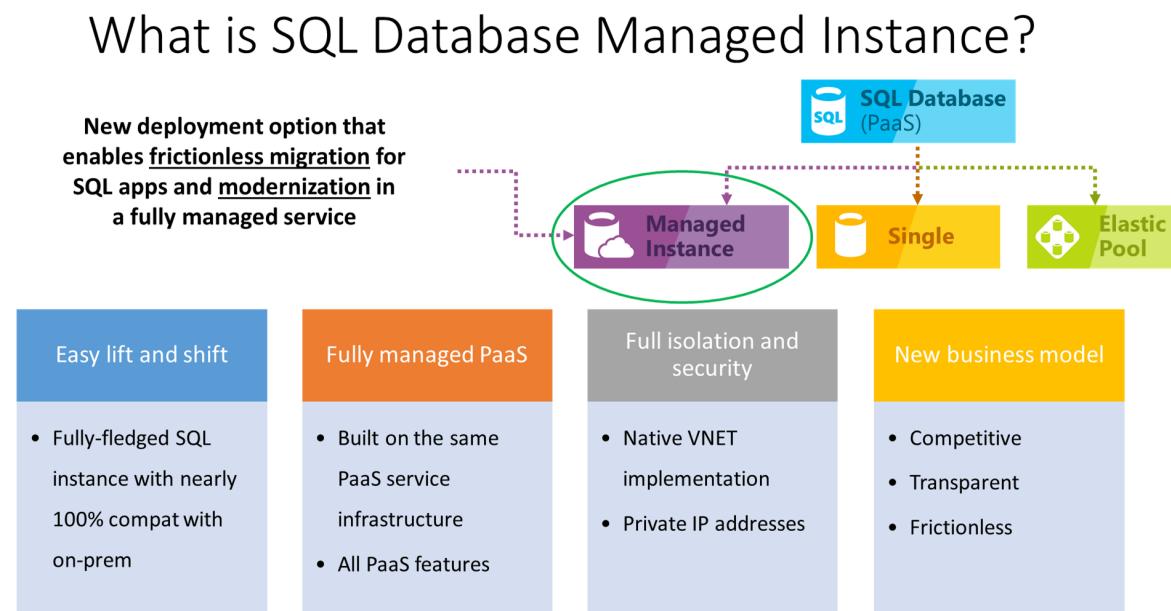
2/13/2020 • 22 minutes to read • [Edit Online](#)

Managed instance is a new deployment option of Azure SQL Database, providing near 100% compatibility with the latest SQL Server on-premises (Enterprise Edition) Database Engine, providing a native [virtual network \(VNet\)](#) implementation that addresses common security concerns, and a [business model](#) favorable for on-premises SQL Server customers. The managed instance deployment model allows existing SQL Server customers to lift and shift their on-premises applications to the cloud with minimal application and database changes. At the same time, the managed instance deployment option preserves all PaaS capabilities (automatic patching and version updates, [automated backups](#), [high-availability](#)), that drastically reduces management overhead and TCO.

## IMPORTANT

For a list of regions in which the managed instance deployment option is currently available, see [supported regions](#).

The following diagram outlines key features of managed instances:



The managed instance deployment model is designed for customers looking to migrate a large number of apps from on-premises or IaaS, self-built, or ISV provided environment to fully managed PaaS cloud environment, with as low migration effort as possible. Using the fully automated [Data Migration Service \(DMS\)](#) in Azure, customers can lift and shift their on-premises SQL Server to a managed instance that offers compatibility with SQL Server on-premises and complete isolation of customer instances with native VNet support. With Software Assurance, you can exchange your existing licenses for discounted rates on a managed instance using the [Azure Hybrid Benefit for SQL Server](#). A managed instance is the best migration destination in the cloud for SQL Server instances that require high security and a rich programmability surface.

The managed instance deployment option aims to deliver close to 100% surface area compatibility with the latest on-premises SQL Server version through a staged release plan.

To decide between the Azure SQL Database deployment options: single database, pooled database, and

managed instance, and SQL Server hosted in virtual machine, see [how to choose the right version of SQL Server in Azure](#).

## Key features and capabilities

Managed instance combines the best features that are available both in Azure SQL Database and SQL Server Database Engine.

### IMPORTANT

A managed instance runs with all of the features of the most recent version of SQL Server, including online operations, automatic plan corrections, and other enterprise performance enhancements. A Comparison of the features available is explained in [Feature comparison: Azure SQL Database versus SQL Server](#).

PAAS BENEFITS	BUSINESS CONTINUITY
No hardware purchasing and management No management overhead for managing underlying infrastructure Quick provisioning and service scaling Automated patching and version upgrade Integration with other PaaS data services	99.99% uptime SLA Built in <a href="#">high-availability</a> Data protected with <a href="#">automated backups</a> Customer configurable backup retention period User-initiated <a href="#">backups</a> <a href="#">Point in time database restore</a> capability
<b>Security and compliance</b>  Isolated environment ( <a href="#">VNet integration</a> , single tenant service, dedicated compute and storage) <a href="#">Transparent data encryption (TDE)</a> <a href="#">Azure AD authentication</a> , single sign-on support <a href="#">Azure AD server principals (logins)</a> Adheres to compliance standards same as Azure SQL database <a href="#">SQL auditing</a> <a href="#">Advanced Threat Protection</a>	<b>Management</b>  Azure Resource Manager API for automating service provisioning and scaling Azure portal functionality for manual service provisioning and scaling Data Migration Service

### IMPORTANT

Azure SQL Database (all deployment options), has been certified against a number of compliance standards. For more information, see the [Microsoft Azure Trust Center](#) where you can find the most current list of SQL Database compliance certifications.

The key features of managed instances are shown in the following table:

FEATURE	DESCRIPTION
SQL Server version / build	SQL Server Database Engine (latest stable)
Managed automated backups	Yes
Built-in instance and database monitoring and metrics	Yes
Automatic software patching	Yes
The latest Database Engine features	Yes

FEATURE	DESCRIPTION
Number of data files (ROWS) per the database	Multiple
Number of log files (LOG) per database	1
VNet - Azure Resource Manager deployment	Yes
VNet - Classic deployment model	No
Portal support	Yes
Built-in Integration Service (SSIS)	No - SSIS is a part of <a href="#">Azure Data Factory PaaS</a>
Built-in Analysis Service (SSAS)	No - SSAS is separate <a href="#">PaaS</a>
Built-in Reporting Service (SSRS)	No - use Power BI or SSRS IaaS

## vCore-based purchasing model

The [vCore-based purchasing model](#) for managed instances gives you flexibility, control, transparency, and a straightforward way to translate on-premises workload requirements to the cloud. This model allows you to change compute, memory, and storage based upon your workload needs. The vCore model is also eligible for up to 55 percent savings with the [Azure Hybrid Benefit for SQL Server](#).

In vCore model, you can choose between generations of hardware.

- **Gen4** Logical CPUs are based on Intel E5-2673 v3 (Haswell) 2.4-GHz processors, attached SSD, physical cores, 7-GB RAM per core, and compute sizes between 8 and 24 vCores.
- **Gen5** Logical CPUs are based on Intel E5-2673 v4 (Broadwell) 2.3-GHz and Intel SP-8160 (Skylake) processors, fast NVMe SSD, hyper-threaded logical core, and compute sizes between 4 and 80 cores.

Find more information about the difference between hardware generations in [managed instance resource limits](#).

## Managed instance service tiers

Managed instance is available in two service tiers:

- **General purpose:** Designed for applications with typical performance and IO latency requirements.
- **Business critical:** Designed for applications with low IO latency requirements and minimal impact of underlying maintenance operations on the workload.

Both service tiers guarantee 99.99% availability and enable you to independently select storage size and compute capacity. For more information on the high availability architecture of Azure SQL Database, see [High availability and Azure SQL Database](#).

### General purpose service tier

The following list describes key characteristic of the General Purpose service tier:

- Design for the majority of business applications with typical performance requirements
- High-performance Azure Blob storage (8 TB)
- Built-in [high-availability](#) based on reliable Azure Blob storage and [Azure Service Fabric](#)

For more information, see [storage layer in general purpose tier](#) and [storage performance best practices and considerations for managed instances \(general purpose\)](#).

Find more information about the difference between service tiers in [managed instance resource limits](#).

### Business Critical service tier

Business Critical service tier is built for applications with high IO requirements. It offers highest resilience to failures using several isolated replicas.

The following list outlines the key characteristics of the Business Critical service tier:

- Designed for business applications with highest performance and HA requirements
- Comes with super-fast local SSD storage (up to 1 TB on Gen4 and up to 4 TB on Gen5)
- Built-in [high availability](#) based on [Always On Availability Groups](#) and [Azure Service Fabric](#).
- Built-in additional [read-only database replica](#) that can be used for reporting and other read-only workloads
- [In-Memory OLTP](#) that can be used for workload with high-performance requirements

Find more information about the difference between service tiers in [managed instance resource limits](#).

## Managed instance management operations

Azure SQL Database provides management operations that you can use to automatically deploy new managed instances, update instance properties, and delete instances when no longer needed. This section provides information about management operations and their typical durations.

To support [deployments within Azure Virtual Networks \(VNets\)](#) and provide isolation and security for customers, managed instance relies on [virtual clusters](#), which represent a dedicated set of isolated virtual machines deployed inside the customer's virtual network subnet. Essentially, every managed instance deployment in an empty subnet results in a new virtual cluster buildout.

Subsequent operations on deployed managed instances might also have effects on its underlying virtual cluster. This affects the duration of management operations, as deploying additional virtual machines comes with an overhead that needs to be considered when you plan new deployments or updates to existing managed instances.

All management operations can be categorized as follows:

- Instance deployment (new instance creation).
- Instance update (changing instance properties, such as vCores or reserved storage).
- Instance deletion.

Typically, operations on virtual clusters take the longest. Duration of the operations on virtual clusters vary – below are the values that you can typically expect, based on existing service telemetry data:

- Virtual cluster creation. This is a synchronous step in instance management operations. **90% of operations finish in 4 hours.**
- Virtual cluster resizing (expansion or shrinking). Expansion is a synchronous step, while shrinking is performed asynchronously (without impact on the duration of instance management operations). **90% of cluster expansions finish in less than 2.5 hours.**
- Virtual cluster deletion. Deletion is an asynchronous step, but it can also be [initiated manually](#) on an empty virtual cluster, in which case it executes synchronously. **90% of virtual cluster deletions finish in 1.5 hours.**

Additionally, management of instances may also include one of the operations on hosted databases, which results in longer durations:

- Attaching database files from Azure Storage. This is a synchronous step, such as compute (vCore), or storage scaling up or down in the General Purpose service tier. **90% of these operations finish in 5 minutes.**
- Always On availability group seeding. This is a synchronous step, such as compute (vCore), or storage scaling in the Business Critical service tier as well as in changing the service tier from General Purpose to Business Critical (or vice versa). Duration of this operation is proportional to the total database size as well as current database activity (number of active transactions). Database activity when updating an instance can introduce significant variance to the total duration. **90% of these operations execute at 220 GB / hour or higher.**

The following table summarizes operations and typical overall durations:

CATEGORY	OPERATION	LONG-RUNNING SEGMENT	ESTIMATED DURATION
<b>Deployment</b>	First instance in an empty subnet	Virtual cluster creation	90% of operations finish in 4 hours
Deployment	First instance of another hardware generation in a non-empty subnet (for example, first Gen 5 instance in a subnet with Gen 4 instances)	Virtual cluster creation*	90% of operations finish in 4 hours
Deployment	First instance creation of 4 vCores, in an empty or non-empty subnet	Virtual cluster creation**	90% of operations finish in 4 hours
Deployment	Subsequent instance creation within the non-empty subnet (2nd, 3rd, etc. instance)	Virtual cluster resizing	90% of operations finish in 2.5 hours
<b>Update</b>	Instance property change (admin password, AAD login, Azure Hybrid Benefit flag)	N/A	Up to 1 minute
Update	Instance storage scaling up/down (General Purpose service tier)	Attaching database files	90% of operations finish in 5 minutes
Update	Instance storage scaling up/down (Business Critical service tier)	- Virtual cluster resizing - Always On availability group seeding	90% of operations finish in 2.5 hours + time to seed all databases (220 GB / hour)
Update	Instance compute (vCores) scaling up and down (General Purpose)	- Virtual cluster resizing - Attaching database files	90% of operations finish in 2.5 hours
Update	Instance compute (vCores) scaling up and down (Business Critical)	- Virtual cluster resizing - Always On availability group seeding	90% of operations finish in 2.5 hours + time to seed all databases (220 GB / hour)

Category	Operation	Long-running segment	Estimated duration
Update	Instance scale down to 4 vCores (General Purpose)	- Virtual cluster resizing (if done for the first time, it may require virtual cluster creation**) - Attaching database files	90% of operations finish in 4 h 5 min**
Update	Instance scale down to 4 vCores (Business Critical)	- Virtual cluster resizing (if done for the first time, it may require virtual cluster creation**) - Always On availability group seeding	90% of operations finish in 4 hours + time to seed all databases (220 GB / hour)
Update	Instance service tier change (General Purpose to Business Critical and vice versa)	- Virtual cluster resizing - Always On availability group seeding	90% of operations finish in 2.5 hours + time to seed all databases (220 GB / hour)
<b>Deletion</b>	Instance deletion	Log tail backup for all databases	90% operations finish in up to 1 minute. Note: if last instance in the subnet is deleted, this operation will schedule virtual cluster deletion after 12 hours***
Deletion	Virtual cluster deletion (as user-initiated operation)	Virtual cluster deletion	90% of operations finish in up to 1.5 hours

\* Virtual cluster is built per hardware generation.

\*\* The 4 vCores deployment option was released in June 2019 and requires a new virtual cluster version. If you had instances in the target subnet that were all created before June 12, a new virtual cluster will be deployed automatically to host 4 vCore instances.

\*\*\* 12 hours is the current configuration but that might change in the future, so don't take a hard dependency on it. If you need to delete a virtual cluster earlier (to release the subnet for example), see [Delete a subnet after deleting an Azure SQL Database managed instance](#).

### Instance availability during management

Managed instances are not available to client applications during deployment and deletion operations.

Managed instances are available during update operations but there is a short downtime caused by the failover that happens at the end of updates that typically lasts up to 10 seconds. The exception to this is update of the reserved storage space in General Purpose service tier which does not incur failover nor affect instance availability.

#### IMPORTANT

Duration of a failover can vary significantly in case of long-running transactions that happen on the databases due to [prolonged recovery time](#). Hence it's not recommended to scale compute or storage of Azure SQL Database managed instance or to change service tier at the same time with the long-running transactions (data import, data processing jobs, index rebuild, etc.). Database failover that will be performed at the end of the operation will cancel ongoing transactions and result in prolonged recovery time.

**TIP**

Update of the reserved storage space in General Purpose service tier does not incur failover nor affect instance availability.

[Accelerated database recovery](#) is not currently available for Azure SQL Database managed instances.

Once enabled, this feature will significantly reduce variability of failover time, even in case of long-running transactions.

### Cancelling management operations

The following table summarizes ability to cancel specific management operations and typical overall durations:

CATEGORY	OPERATION	CANCELABLE	ESTIMATED CANCEL DURATION
Deployment	Instance creation	No	
Update	Instance storage scaling up/down (General Purpose)	No	
Update	Instance storage scaling up/down (Business Critical)	Yes	90% of operations finish in 5 minutes
Update	Instance compute (vCores) scaling up and down (General Purpose)	Yes	90% of operations finish in 5 minutes
Update	Instance compute (vCores) scaling up and down (Business Critical)	Yes	90% of operations finish in 5 minutes
Update	Instance service tier change (General Purpose to Business Critical and vice versa)	Yes	90% of operations finish in 5 minutes
Delete	Instance deletion	No	
Delete	Virtual cluster deletion (as user-initiated operation)	No	

In order to cancel the management operation, go to the overview blade and click on notification box of ongoing operation. From the right side, a screen with ongoing operation will appear and there will be button for canceling operation. After first click, you will be asked to click again and confirm that you want to cancel the operation.

Pricing tier update is in progress. Please do not create, delete, restore databases, or modify the pricing tier options until the ongoing operation is completed.

**Resource group (change) :** Managed instance admin : urmila

Status : Updating Host :

Location : West Europe Pricing tier : General Purpose Gen5 (256 GB, 4 vCores)

Subscription (change) : Instance pool : Not in an instance pool

Subscription ID : Virtual network/subnet : MIVirtualNetwork/ManagedInsanceSubnet

Tags (change) : Click here to add tags Virtual cluster :

**CPU utilization**

1 hour 24 hours 7 days Aggregation type: Avg

Storage utilization

Current 1 GB Quick 256 GB

**4 Managed Instance databases**

Notifications (1) Managed Instance features (1)

All Alerts (0) Recommendations (0) Info (1)

You have 1 ongoing operation

This managed instance has some ongoing or recent operations. Click here to view them.

After cancel request has been submitted and processed, you will get notification if cancel submission has been successful or not.

In case of cancel success, management operation will be canceled in couple of minutes resulting with a failure.

More events in the activity log → Dismiss all ▾

**! Failed to scale managed instance**

Failed to scale the managed instance:  
ErrorCode: OperationCancelled  
ErrorMessage: The operation has been cancelled by user.  
an hour ago

**✓ Successfully submitted managed instance operation cancellation...**

Successfully submitted managed instance operation cancellation request for management operation on managed instance:  
an hour ago

If cancel request fails or cancel button is not active, that means that management operation has entered not cancelable state and that it will finish in couple of minutes. Management operation will continue its execution until it is completed.

#### IMPORTANT

Canceling operation is currently supported only in Portal.

## Advanced security and compliance

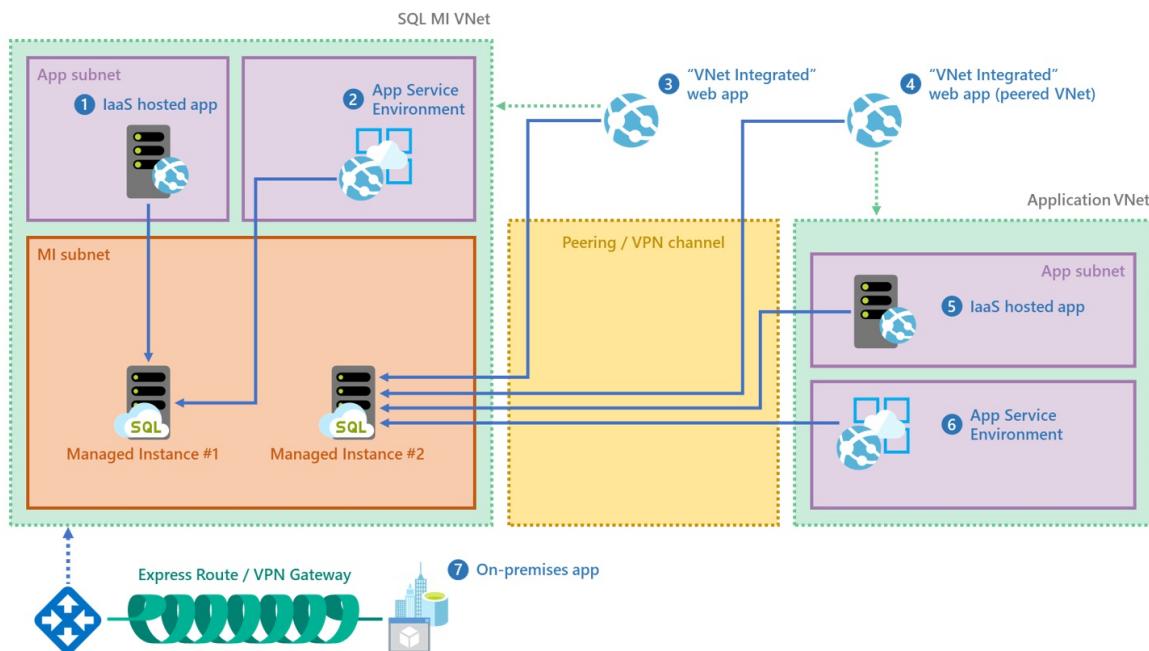
The managed instance deployment option combines advanced security features provided by Azure cloud and SQL Server Database Engine.

## Managed instance security isolation

A managed instance provides additional security isolation from other tenants in the Azure cloud. Security isolation includes:

- [Native virtual network implementation](#) and connectivity to your on-premises environment using Azure Express Route or VPN Gateway.
- In a default deployment, SQL endpoint is exposed only through a private IP address, allowing safe connectivity from private Azure or hybrid networks.
- Single-tenant with dedicated underlying infrastructure (compute, storage).

The following diagram outlines various connectivity options for your applications:



To learn more details about VNet integration and networking policy enforcement at the subnet level, see [VNet architecture for managed instances](#) and [Connect your application to a managed instance](#).

### IMPORTANT

Place multiple managed instance in the same subnet, wherever that is allowed by your security requirements, as that will bring you additional benefits. Collocating instances in the same subnet will significantly simplify networking infrastructure maintenance and reduce instance provisioning time, since long provisioning duration is associated with the cost of deploying the first managed instance in a subnet.

## Azure SQL Database Security Features

Azure SQL Database provides a set of advanced security features that can be used to protect your data.

- [Managed instance auditing](#) tracks database events and writes them to an audit log file placed in your Azure storage account. Auditing can help maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations.
- Data encryption in motion - a managed instance secures your data by providing encryption for data in motion using Transport Layer Security. In addition to transport layer security, the managed instance deployment option offers protection of sensitive data in flight, at rest and during query processing with [Always Encrypted](#). Always Encrypted is an industry-first that offers unparalleled data security against breaches involving the theft of critical data. For example, with Always Encrypted, credit card numbers are stored encrypted in the database always, even during query processing, allowing decryption at the point of use by authorized staff or applications that need to process that data.

- [Advanced Threat Protection](#) complements [auditing](#) by providing an additional layer of security intelligence built into the service that detects unusual and potentially harmful attempts to access or exploit databases. You are alerted about suspicious activities, potential vulnerabilities, and SQL injection attacks, as well as anomalous database access patterns. Advanced Threat Protection alerts can be viewed from [Azure Security Center](#) and provide details of suspicious activity and recommend action on how to investigate and mitigate the threat.
- [Dynamic data masking](#) limits sensitive data exposure by masking it to non-privileged users. Dynamic data masking helps prevent unauthorized access to sensitive data by enabling you to designate how much of the sensitive data to reveal with minimal impact on the application layer. It's a policy-based security feature that hides the sensitive data in the result set of a query over designated database fields, while the data in the database is not changed.
- [Row-level security](#) enables you to control access to rows in a database table based on the characteristics of the user executing a query (such as by group membership or execution context). Row-level security (RLS) simplifies the design and coding of security in your application. RLS enables you to implement restrictions on data row access. For example, ensuring that workers can access only the data rows that are pertinent to their department, or restricting a data access to only the relevant data.
- [Transparent data encryption \(TDE\)](#) encrypts managed instance data files, known as encrypting data at rest. TDE performs real-time I/O encryption and decryption of the data and log files. The encryption uses a database encryption key (DEK), which is stored in the database boot record for availability during recovery. You can protect all your databases in a managed instance with transparent data encryption. TDE is SQL Server's proven encryption-at-rest technology that is required by many compliance standards to protect against theft of storage media.

Migration of an encrypted database to a managed instance is supported via the Azure Database Migration Service (DMS) or native restore. If you plan to migrate an encrypted database using native restore, migration of the existing TDE certificate from the SQL Server on-premises or SQL Server in a virtual machine to a managed instance is a required step. For more information about migration options, see [SQL Server instance migration to managed instance](#).

## Azure Active Directory Integration

The managed instance deployment option supports traditional SQL server Database engine logins and logins integrated with Azure Active Directory (AAD). Azure AD server principals (logins) ([public preview](#)) are Azure cloud version of on-premises database logins that you are using in your on-premises environment. Azure AD server principals (logins) enable you to specify users and groups from your Azure Active Directory tenant as true instance-scoped principals, capable of performing any instance-level operation, including cross-database queries within the same managed instance.

A new syntax is introduced to create Azure AD server principals (logins), **FROM EXTERNAL PROVIDER**. For more information on the syntax, see [CREATE LOGIN](#), and review the [Provision an Azure Active Directory administrator for your managed instance](#) article.

### Azure Active Directory integration and multi-factor authentication

The managed instance deployment option enables you to centrally manage identities of database user and other Microsoft services with [Azure Active Directory integration](#). This capability simplified permission management and enhances security. Azure Active Directory supports [multi-factor authentication](#) (MFA) to increase data and application security while supporting a single sign-on process.

### Authentication

Managed instance authentication refers to how users prove their identity when connecting to the database. SQL Database supports two types of authentication:

- **SQL Authentication:**

This authentication method uses a username and password.

- **Azure Active Directory Authentication:**

This authentication method uses identities managed by Azure Active Directory and is supported for managed and integrated domains. Use Active Directory authentication (integrated security) whenever possible.

## Authorization

Authorization refers to what a user can do within an Azure SQL Database, and is controlled by your user account's database role memberships and object-level permissions. A Managed instance has same authorization capabilities as SQL Server 2017.

## Database migration

The managed instance deployment option targets user scenarios with mass database migration from on-premises or IaaS database implementations. Managed instance supports several database migration options:

### Back up and restore

The migration approach leverages SQL backups to Azure Blob storage. Backups stored in Azure storage blob can be directly restored into a managed instance using the [T-SQL RESTORE command](#).

- For a quickstart showing how to restore the Wide World Importers - Standard database backup file, see [Restore a backup file to a managed instance](#). This quickstart shows you have to upload a backup file to Azure blob storage and secure it using a Shared access signature (SAS) key.
- For information about restore from URL, see [Native RESTORE from URL](#).

#### IMPORTANT

Backups from a managed instance can only be restored to another managed instance. They cannot be restored to an on-premises SQL Server or to a single database/elastic pool.

## Data Migration Service

The Azure Database Migration Service is a fully managed service designed to enable seamless migrations from multiple database sources to Azure Data platforms with minimal downtime. This service streamlines the tasks required to move existing third party and SQL Server databases to Azure SQL Database (single databases, pooled databases in elastic pools, and instance databases in a managed instance) and SQL Server in Azure VM. See [How to migrate your on-premises database to managed instance using DMS](#).

## SQL features supported

The managed instance deployment option aims to deliver close to 100% surface area compatibility with on-premises SQL Server coming in stages until service general availability. For a features and comparison list, see [SQL Database feature comparison](#), and for a list of T-SQL differences in managed instances versus SQL Server, see [managed instance T-SQL differences from SQL Server](#).

The managed instance deployment option supports backward compatibility to SQL 2008 databases. Direct migration from SQL 2005 database servers is supported, compatibility level for migrated SQL 2005 databases are updated to SQL 2008.

The following diagram outlines surface area compatibility in managed instance:

# Easy migration: nearly 100% like SQL Server

<b>Data migration</b> <ul style="list-style-type: none"><li>Native backup/restore</li><li>Configurable DB file layout</li><li>DMS (migrations at scale)</li></ul>	<b>Security</b> <ul style="list-style-type: none"><li>Integrated Auth (Azure AD)</li><li>Encryption (TDE, AE)</li></ul>	<ul style="list-style-type: none"><li>SQL Audit</li><li>Row-Level Security</li><li>Dynamic Data Masking</li></ul>
<b>Programmability</b> <ul style="list-style-type: none"><li>Global temp tables</li><li>Cross-database queries and transactions</li><li>Linked servers</li><li>CLR modules</li></ul>	<b>Operational</b> <ul style="list-style-type: none"><li>DMVs &amp; XEvents</li><li>Query Store</li><li>SQL Agent</li><li>DB Mail (external SMTP)</li></ul>	<b>Scenario enablers</b> <ul style="list-style-type: none"><li>Service Broker</li><li>Change Data Capture</li><li>Transactional Replication</li></ul>

## Key differences between SQL Server on-premises and in a managed instance

The managed instance deployment option benefits from being always-up-to-date in the cloud, which means that some features in on-premises SQL Server may be either obsolete, retired, or have alternatives. There are specific cases when tools need to recognize that a particular feature works in a slightly different way or that the service is running in an environment you do not fully control:

- High-availability is built in and pre-configured using technology similar to [Always On Availability Groups](#).
- Automated backups and point in time restore. Customer can initiate `copy-only` backups that do not interfere with automatic backup chain.
- Managed instance does not allow specifying full physical paths so all corresponding scenarios have to be supported differently: RESTORE DB does not support WITH MOVE, CREATE DB doesn't allow physical paths, BULK INSERT works with Azure Blobs only, etc.
- Managed instance supports [Azure AD authentication](#) as cloud alternative to Windows authentication.
- Managed instance automatically manages XTP filegroup and files for databases containing In-Memory OLTP objects
- Managed instance supports SQL Server Integration Services (SSIS) and can host SSIS catalog (SSISDB) that stores SSIS packages, but they are executed on a managed Azure-SSIS Integration Runtime (IR) in Azure Data Factory (ADF), see [Create Azure-SSIS IR in ADF](#). To compare the SSIS features in SQL Database, see [Compare an Azure SQL Database single database, elastic pool, and managed instance](#).

## Managed instance administration features

The managed instance deployment option enables system administrator to spend less time on administrative tasks because the SQL Database service either performs them for you or greatly simplifies those tasks. For example, [OS / RDBMS installation and patching](#), [dynamic instance resizing and configuration](#), [backups](#), [database replication](#) (including system databases), [high availability configuration](#), and configuration of health and [performance monitoring](#) data streams.

### IMPORTANT

For a list of supported, partially supported, and unsupported features, see [SQL Database features](#). For a list of T-SQL differences in managed instances versus SQL Server, see [managed instance T-SQL differences from SQL Server](#)

## How to programmatically identify a managed instance

The following table shows several properties, accessible through Transact SQL, that you can use to detect

that your application is working with managed instance and retrieve important properties.

PROPERTY	VALUE	COMMENT
<code>@@VERSION</code>	Microsoft SQL Azure (RTM) - 12.0.2000.8 2018-03-07 Copyright (C) 2018 Microsoft Corporation.	This value is same as in SQL Database. This <b>does not</b> indicate SQL engine version 12 (SQL Server 2014). Managed instance always runs the latest stable SQL engine version, which is equal to or higher than latest available RTM version of SQL Server.
<code>SERVERPROPERTY ('Edition')</code>	SQL Azure	This value is same as in SQL Database.
<code>SERVERPROPERTY('EngineEdition')</code>	8	This value uniquely identifies a managed instance.
<code>@@SERVERNAME , SERVERPROPERTY ('ServerName')</code>	Full instance DNS name in the following format: <code>&lt;instanceName&gt;. &lt;dnsPrefix&gt;.database.windows.net</code> , where <code>&lt;instanceName&gt;</code> is name provided by the customer, while <code>&lt;dnsPrefix&gt;</code> is autogenerated part of the name guaranteeing global DNS name uniqueness ("wcus17662feb9ce98", for example)	Example: my-managed-instance.wcus17662feb9ce98.database.windows.net

## Next steps

- To learn how to create your first managed instance, see [Quickstart guide](#).
- For a features and comparison list, see [SQL common features](#).
- For more information about VNet configuration, see [managed instance VNet configuration](#).
- For a quickstart that creates a managed instance and restores a database from a backup file, see [create a managed instance](#).
- For a tutorial using the Azure Database Migration Service (DMS) for migration, see [managed instance migration using DMS](#).
- For advanced monitoring of managed instance database performance with built-in troubleshooting intelligence, see [Monitor Azure SQL Database using Azure SQL Analytics](#).
- For pricing information, see [SQL Database managed instance pricing](#).

# What are SQL Database instance pools (preview)?

2/24/2020 • 8 minutes to read • [Edit Online](#)

Instance pools are a new resource in Azure SQL Database that provides a convenient and cost-efficient way to migrate smaller SQL instances to the cloud at scale.

Instance pools allow you to pre-provision compute resources according to your total migration requirements. You can then deploy several individual managed instances up to your pre-provisioned compute level. For example, if you pre-provision 8 vCores you can deploy two 2 vCore and one 4 vCore instances, and then migrate databases into these instances. Prior to instance pools being available, smaller and less compute-intensive workloads would often have to be consolidated into a larger managed instance when migrating to the cloud. The need to migrate groups of databases to a large instance typically required careful capacity planning and resource governance, additional security considerations, and some extra data consolidation work at the instance level.

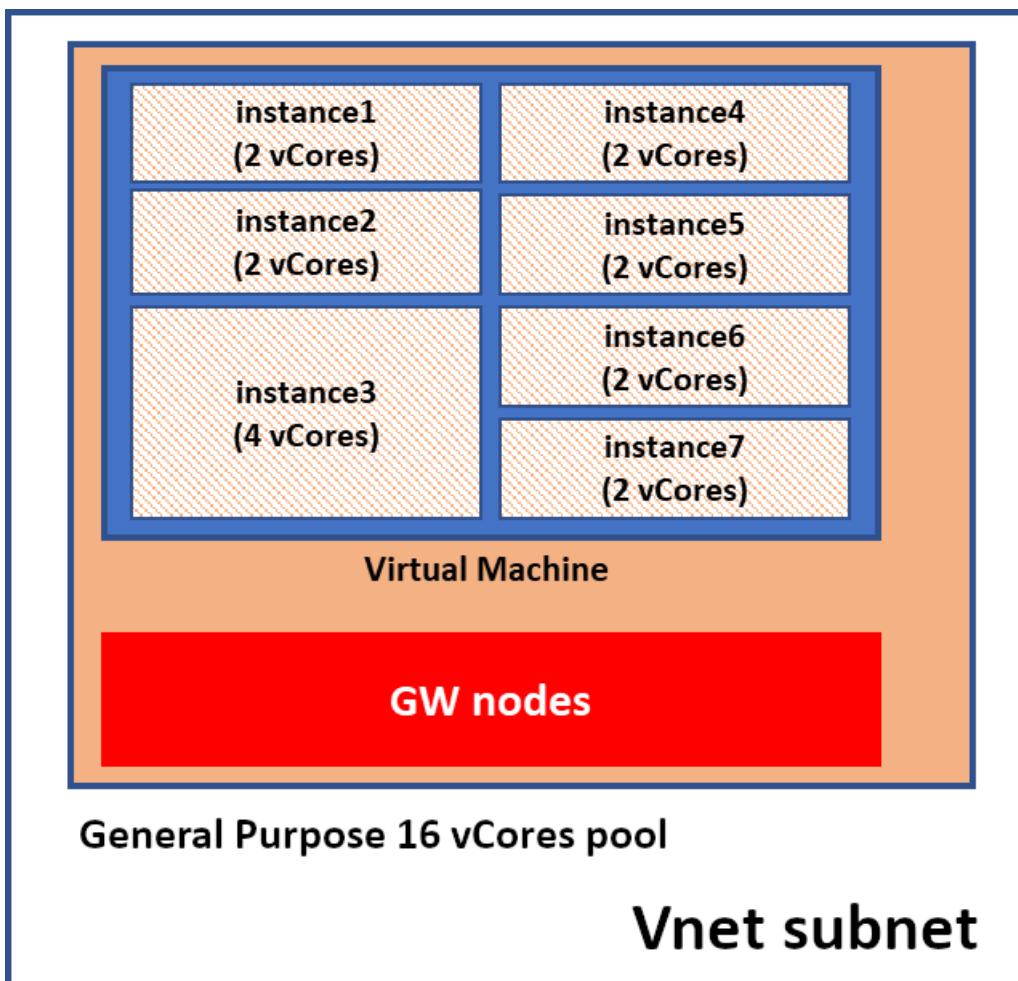
Additionally, instance pools support native VNet integration so you can deploy multiple instance pools and multiple single instances in the same subnet.

## Key capabilities of instance pools

Instance pools provide the following benefits:

1. Ability to host 2 vCore instances. *\*Only for instances in instance pools.*
2. Predictable and fast instance deployment time (up to 5 minutes).
3. Minimal IP address allocation.

The following diagram illustrates an instance pool with multiple instances deployed within a virtual network subnet.



Instance pools enable deployment of multiple instances on the same virtual machine where the virtual machine's compute size is based on the total number of vCores allocated for the pool. This architecture allows *partitioning* of the virtual machine into multiple instances, which can be any supported size, including 2 vCores (2 vCore instances are only available for instances in pools).

Management operations on instances in a pool are much faster once the pool is initially deployed. These operations are faster because deployment or extension of a [virtual cluster](#) (dedicated set of virtual machines) is not part of provisioning the managed instance.

Because all instances in a pool share the same virtual machine, the total IP allocation does not depend on the number of instances deployed, which is convenient for deployment in subnets with a narrow IP range.

Each pool has a fixed IP allocation of only nine IP addresses (not including the five IP addresses in the subnet that are reserved for its own needs). For details, see [subnet size requirements for single instances](#).

## Application scenarios for instance pools

The following list provides the main use cases where instance pools should be considered:

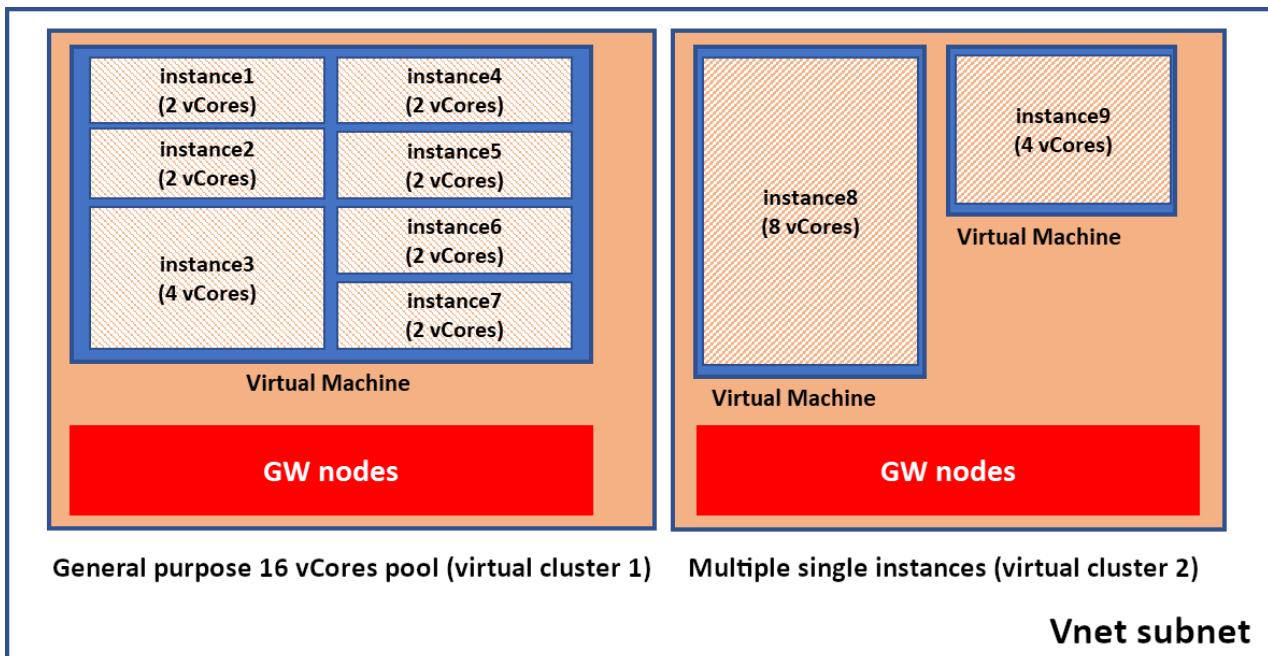
- Migration of a *group of SQL instances* at the same time, where the majority is a smaller size (for example 2 or 4 vCores).
- Scenarios where *predictable and short instance creation or scaling* is important. For example, deployment of a new tenant in a multi-tenant SaaS application environment that requires instance-level capabilities.
- Scenarios where having a *fixed cost or spending limit* is important. For example, running shared dev-test or demo environments of a fixed (or infrequently changing) size, where you periodically deploy managed instances when needed.
- Scenarios where *minimal IP address allocation* in a VNet subnet is important. All instances in a pool are sharing a virtual machine, so the number of allocated IP addresses is lower than in the case of single instances.

# Architecture of instance pools

Instance pools have similar architecture to regular managed instances (*single instances*). To support [deployments within Azure Virtual Networks \(VNets\)](#) and to provide isolation and security for customers, instance pools also rely on [virtual clusters](#). Virtual clusters represent a dedicated set of isolated virtual machines deployed inside the customer's virtual network subnet.

The main difference between the two deployment models is that instance pools allow multiple SQL Server process deployments on the same virtual machine node, which are resource governed using [Windows Job Objects](#), while single instances are always alone on a virtual machine node.

The following diagram shows an instance pool and two individual instances deployed in the same subnet and illustrates the main architectural details for both deployment models:



Every instance pool creates a separate virtual cluster underneath. Instances within a pool and single instances deployed in the same subnet do not share compute resources allocated to SQL Server processes and gateway components, this ensures performance predictability.

## Instance pools resource limitations

There are several resource limitations regarding instance pools and instances inside pools:

- Instance pools are available only on Gen5 hardware.
- Instances within a pool have dedicated CPU and RAM, so the aggregated number of vCores across all instances must be less than or equal to the number of vCores allocated to the pool.
- All [instance level limits](#) apply to instances created within a pool.
- In addition to instance-level limits there are also two limits imposed *at the instance pool level*:
  - Total storage size per pool (8 TB).
  - Total number of databases per pool (100).

Total storage allocation and number of databases across all instances must be lower or equal to the limits exposed by instance pools.

- Instance pools support 8, 16, 24, 32, 40, 64, and 80 vCores.
- Managed instances inside pools support 2, 4, 8, 16, 24, 32, 40, 64 and 80 vCores.
- Managed instances inside pools support storage sizes between 32 GB and 8 TB, except:
  - 2 vCore instances support sizes between 32 GB and 640 GB

- 4 vCore instances support sizes between 32 GB and 2 TB

The [service tier property](#) is associated with the instance pool resource so all instances in a pool must be the same service tier as the service tier of the pool. At this time, only the General Purpose service tier is available (see the following section on limitations in the current preview).

### Public preview limitations

The public preview has the following limitations:

- Currently, only the General Purpose service tier is available.
- Instance pools cannot be scaled during the public preview so careful capacity planning before deployment is important.
- Azure portal support for instance pool creation and configuration is not yet available. All operations on instance pools are supported through PowerShell only. Initial instance deployment in a pre-created pool is also supported through PowerShell only. Once deployed into a pool, managed instances can be updated using the Azure portal.
- Managed instances created outside of the pool cannot be moved into an existing pool and instances created inside a pool cannot be moved outside as a single instance or to another pool.
- Reserved instance pricing (license included or with Azure Hybrid Benefit) is not available.

## SQL features supported

Instances created in pools support the same [compatibility levels and features supported in single managed instances](#).

Every managed instance deployed in a pool has a separate instance of SQL Agent.

Optional features or features that require you to choose specific values (such as instance-level collation, time zone, public endpoint for data traffic, failover groups) are configured at instance level and can be different for each instance in a pool.

## Performance considerations

Although managed instances within pools do have dedicated vCore and RAM, they share local disk (for tempdb usage) and network resources. It's not likely, but it is possible to experience the *noisy neighbor* effect if multiple instances in the pool have high resource consumption at the same time. If you observe this behavior, consider deploying these instances to a bigger pool or as single instances.

## Security considerations

Because instances deployed in a pool share the same virtual machine, you may want to consider disabling features that introduce higher security risks, or to firmly control access permissions to these features. For example, CLR integration, native backup and restore, database email, etc.

## Instance pool support requests

Create and manage support requests for instance pools in the [Azure portal](#).

If you are experiencing issues related to instance pool deployment (creation or deletion), make sure that you specify **Instance Pools** in the **Problem subtype** field.

Dashboard > Help + support - New support request

## Help + support - New support request

Search (Ctrl+ /)

Basics Solutions Details Review + create

Create a new support request to get assistance with billing, subscription, technical (including advisory) or quota management issues. Complete the Basics tab by selecting the options that best describe your problem. Providing detailed, accurate information can help to solve your issues faster.

\* Issue type: Technical

\* Subscription: Can't find your subscription? Show more ⓘ

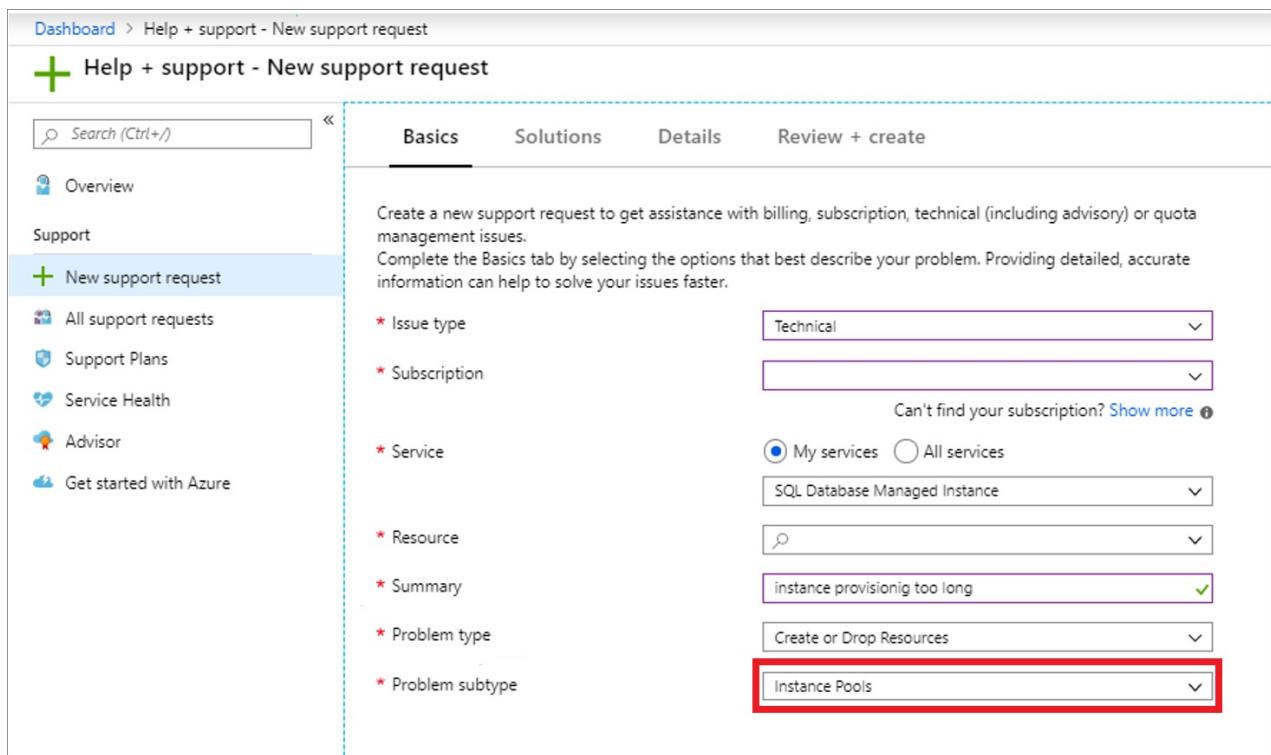
\* Service: My services (selected) All services

\* Resource: SQL Database Managed Instance

\* Summary: instance provisioning too long

\* Problem type: Create or Drop Resources

\* Problem subtype: Instance Pools (highlighted with a red box)



If you are experiencing issues related to single instances or databases within a pool, you should create a regular support ticket for Azure SQL Database managed instances.

To create larger managed instance deployments (with or without instance pools), you may need to obtain a larger regional quota. For more information, see [Request quota increases for Azure SQL Database](#). Note that if you are using instance pools, the deployment logic compares total vCore consumption *at the pool level* against your quota to determine whether you are allowed to create new resources without further increasing your quota.

## Instance pool billing

Instance pools allow scaling compute and storage independently. Customers pay for compute associated with the pool resource measured in vCores, and storage associated with every instance measured in gigabytes (the first 32 GB are free of charge for every instance).

vCore price for a pool is charged regardless of how many instances are deployed in that pool.

For the Compute price (measured in vCores), two pricing options are available:

1. *License included*: Price of SQL licenses is included. This is for the customers who choose not to apply existing SQL Server licenses with Software Assurance.
2. *Azure Hybrid Benefit*: A reduced price that includes Azure Hybrid Benefit for SQL Server. Customers can opt into this price by using their existing SQL Server licenses with Software Assurance. For eligibility and other details, see [Azure Hybrid Benefit](#).

Setting different pricing options is not possible for individual instances in a pool. All instances in the parent pool must be either at License Included price or Azure Hybrid Benefit price. The license model for the pool can be altered after the pool is created.

### IMPORTANT

If you specify a License Model for the instance that is different than in the pool, the pool price is used and the instance level value is ignored.

If you create instance pools on [subscriptions eligible for dev-test benefit](#), you automatically receive discounted rates of up to 55 percent on Azure SQL managed instance.

For full details on instance pool pricing, refer to the *instance pools* section on the [managed instance pricing page](#).

## Next steps

- To get started with instance pools, see [SQL Database instance pools how-to guide](#).
- To learn how to create your first managed instance, see [Quickstart guide](#).
- For a features and comparison list, see [SQL common features](#).
- For more information about VNet configuration, see [managed instance VNet configuration](#).
- For a quickstart that creates a managed instance and restores a database from a backup file, see [create a managed instance](#).
- For a tutorial using the Azure Database Migration Service (DMS) for migration, see [managed instance migration using DMS](#).
- For advanced monitoring of managed instance database performance with built-in troubleshooting intelligence, see [Monitor Azure SQL Database using Azure SQL Analytics](#).
- For pricing information, see [SQL Database managed instance pricing](#).

# SQL Database managed instance frequently asked questions (FAQ)

2/13/2020 • 8 minutes to read • [Edit Online](#)

This article contains many of the most common questions about [SQL Database managed instance](#).

## Supported features

### **Where can I find a list of features supported on managed instance?**

For a list of supported features in managed instance, see [Azure SQL Database versus SQL Server](#).

For differences in syntax and behavior between Azure SQL Database managed instance and on-premises SQL Server, see [T-SQL differences from SQL Server](#).

## Tech spec & resource limits

### **Where can I find technical characteristics and resource limits for managed instance?**

For available hardware generation characteristics, see [technical differences in hardware generations](#). For available service tiers and their characteristics, see [technical differences between service tiers](#).

## Known issues & bugs

### **Where can I find known issues and bugs?**

For bugs and known issues see [known issues](#).

## New features

### **Where can I find latest features and the features in public preview?**

For new and preview features see [release notes](#).

## Deployment times

### **How much time takes to create or update instance, or to restore a database?**

Expected time to create new managed instance or change service tier (vCores, storage) depend on several factors. Take a look at the [Management operations](#)

## Naming convention

### **Can a managed instance have the same name as on-premises SQL Server?**

Managed instance must have a name that ends with *database.windows.net*. To use another DNS zone instead of the default, for example, **mi-another-name**.contoso.com:

- Use CliConfig to define an alias. The tool is just a registry settings wrapper, so it could be done using group policy or script as well.
- Use *CNAME* with *TrustServerCertificate=true* option.

## Move DB from MI

### How can I move database from managed instance back to SQL Server or Azure SQL Database?

You can [export database to BACPAC](#) and then [import the BACPAC file](#). This is a recommended approach if your database is smaller than 100 GB.

Transactional replication can be used if all tables in the database have primary keys.

Native `COPY_ONLY` backups taken from managed instance cannot be restored to SQL Server because managed instance has a higher database version compared to SQL Server.

## Migrate instance DB

### How can I migrate my instance database to a single Azure SQL Database?

One option is to [export the database to a BACPAC](#) and then [import the BACPAC file](#).

This is the recommended approach if your database is smaller than 100 GB. Transactional replication can be used if all tables in the database have primary keys.

## Switch hardware generation

### Can I switch my managed instance hardware generation between Gen 4 and Gen 5 online?

Automated online switching between hardware generations is possible if both hardware generations are available in the region where your managed instance is provisioned. In this case, you can check [vCore model overview page](#) explaining how to switch between hardware generations.

This is a long-running operation as a new managed instance will be provisioned in the background and databases automatically transferred between the old and new instance with a quick failover at the end of the process.

If both hardware generations are not supported in the same region, changing the hardware generation is possible but must be done manually. This requires you to provision a new instance in the region where the wanted hardware generation is available, and manually back up and restore data between the old and new instance.

## Tune performance

### How do I tune performance of my managed instance?

General Purpose managed instance uses remote storage due to which size of data and log files matters to performance. For more information, see [Impact of log file size on General Purpose Managed Instance performance](#).

If your workload consists of lots of small transactions, consider switching the connection type from proxy to redirect mode.

## Maximum storage size

### What is the maximum storage size for managed instance?

Storage size for managed instance depends on the selected service tier (General Purpose or Business Critical). For storage limitations of these service tiers, see [Service tier characteristic](#).

## Back up storage cost

### Is the backup storage deducted from my managed instance storage?

No, backup storage is not deducted from your managed instance storage space. The backup storage is independent from the instance storage space and it is not limited in size. Backup storage is limited by the time period to retain the backup of your instance databases, configurable from 7 to 35 days. For details, see [Automated backups](#).

## Track billing

### Is there a way to track my billing cost for my managed instance?

You can do so using the [Azure Cost Management solution](#). Navigate to **Subscriptions** in the [Azure portal](#) and select **Cost Analysis**.

Use the **Accumulated costs** option and then filter by the **Resource type** as `microsoft.sql/managedinstances`.

## Inbound NSG rules

### How can I set inbound NSG rules on management ports?

The built-in firewall feature configures Windows firewall on all virtual machines in the cluster to allow inbound connections from IP ranges associated only to Microsoft management/deployment machines and secure admin workstations effectively preventing intrusions through the network layer.

Here is what ports are used for:

Ports 9000 and 9003 are used by Service Fabric infrastructure. Service Fabric primary role is to keep the virtual cluster healthy and keep goal state in terms of number of component replicas.

Ports 1438, 1440, and 1452 are used by Node Agent. Node agent is an application that runs inside the cluster and is used by the control plane to execute management commands.

In addition to the built-in firewall on the network layer, communication is also protected with certificates. For more information and how to verify the built-in firewall, see [Azure SQL Database managed instance built-in firewall](#).

## Mitigate network risks

### How can I mitigate networking risks?

To mitigate any networking risks, customers are recommended to apply a set of security settings and controls:

- Turn on [Transparent Data Encryption \(TDE\)](#) on all databases.
- Turn off Common Language Runtime (CLR). This is recommended on-premises as well.
- Use Azure Active Directory (AAD) authentication only.
- Access instance with low privileged DBA account.
- Configure JiT jumpbox access for sysadmin account.
- Turn on [SQL auditing](#), and integrate it with alerting mechanisms.
- Turn on the [Threat Detection](#) from the [Advanced Data Security \(ADS\)](#) suite.

## Cost saving use cases

### Where can I find use cases and resulting cost savings with managed instance?

Managed instance case studies:

- [Komatsu](#)
- [KMD](#)
- [PowerDETAILS](#)
- [Allscripts](#) To get a better understanding of the benefits, costs, and risks associated with deploying Azure SQL

Database managed instance, there's also a Forrester's study: [Total Economic Impact of MI](#).

## DNS Refresh

### **Can I do DNS refresh?**

Currently, we don't provide a feature to refresh DNS server configuration for managed instance.

DNS configuration is eventually refreshed:

- When DHCP lease expires.
- On platform upgrade.

As a workaround, downgrade the managed instance to 4 vCore and upgrade it again afterward. This has a side effect of refreshing the DNS configuration.

## Static IP address

### **Can a managed instance have a static IP address?**

In rare but necessary situations, we might need to do an online migration of a managed instance to a new virtual cluster. If needed, this migration is because of changes in our technology stack aimed to improve security and reliability of the service. Migrating to a new virtual cluster results in changing the IP address that is mapped to the managed instance host name. The managed instance service doesn't claim static IP address support and reserves the right to change it without notice as a part of regular maintenance cycles.

For this reason, we strongly discourage relying on immutability of the IP address as it could cause unnecessary downtime.

## Moving MI

### **Can I move a managed instance or its VNet to another resource group?**

No, this is current platform limitation. After a managed instance is created, moving the managed instance or VNet to another resource group or subscription is not supported.

## Change time zone

### **Can I change the time zone for an existing managed instance?**

Time zone configuration can be set when a managed instance is provisioned for the first time. Changing the time zone of the existing managed instance isn't supported. For details, see [time zone limitations](#).

Workarounds include creating a new managed instance with the proper time zone and then either perform a manual backup and restore, or what we recommend, perform a [cross-instance point-in-time restore](#).

## Resolve performance issues

### **How do I resolve performance issues with my managed instance?**

For a performance comparison between managed instance and SQL Server, a good starting point is [Best practices for performance comparison between Azure SQL managed instance and SQL Server](#) article.

Data loading is often slower on managed instance than in SQL Server due to mandatory full recovery model and [limits](#) on transaction log write throughput. Sometimes, this can be worked around by loading transient data into tempdb instead of user database, or using clustered columnstore, or memory-optimized tables.

## Restore encrypted backup

### Can I restore my encrypted database to managed instance?

Yes, you don't need to decrypt your database to be able to restore it to managed instance. You do need to provide a certificate/key used as an encryption key protector in the source system to the managed instance to be able to read data from the encrypted backup file. There are two possible ways to do it:

- *Upload certificate-protector to the managed instance.* It can be done using PowerShell only. The [sample script](#) describes the whole process.
- *Upload asymmetric key-protector to Azure Key Vault (AKV) and point managed instance to it.* This approach resembles bring-your-own-key (BYOK) TDE use case that also uses AKV integration to store the encryption key. If you don't want to use the key as an encryption key protector, and just want to make the key available for managed instance to restore encrypted database(s), follow instructions for [setting up BYOK TDE](#), and don't check the checkbox *Make the selected key the default TDE protector*.

Once you make the encryption protector available to managed instance, you can proceed with the standard database restore procedure.

## Migrate from single DB

### How can I migrate from Azure SQL Database single or elastic pool to managed instance?

Managed instance offers the same performance levels per compute and storage size as other deployment options of Azure SQL Database. If you want to consolidate data on a single instance, or you simply need a feature supported exclusively in managed instance, you can migrate your data by using export/import (BACPAC) functionality.

# Quickstart: Create an Azure SQL Database managed instance

11/8/2019 • 6 minutes to read • [Edit Online](#)

This quickstart walks you through how to create an Azure SQL Database [managed instance](#) in Azure portal.

## IMPORTANT

For limitations, see [Supported regions](#) and [Supported subscription types](#).

## Sign in to Azure portal

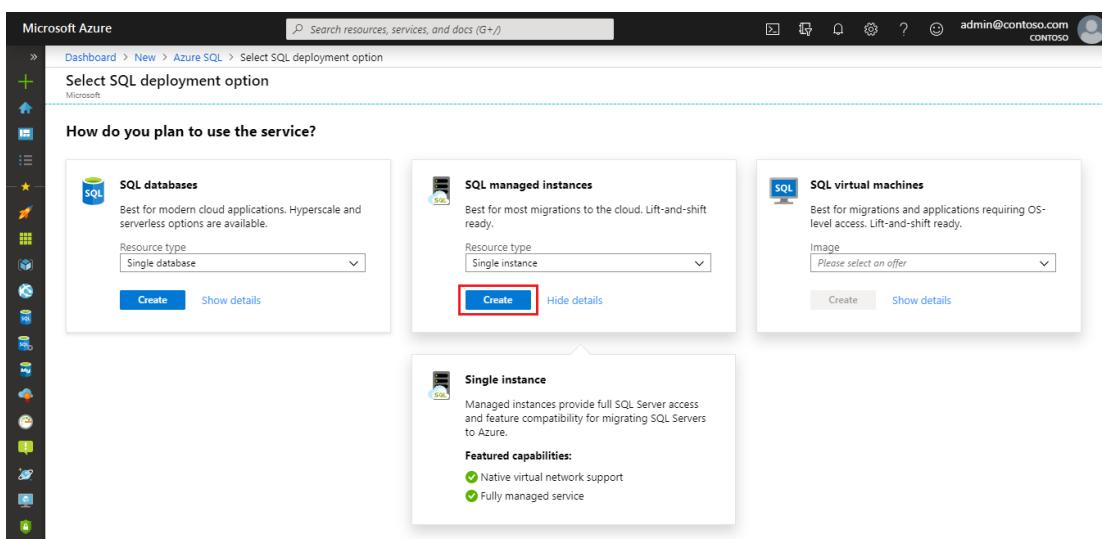
If you don't have an Azure subscription, [create a free account](#).

Sign in to [Azure portal](#).

## Create a managed instance

The following steps show you how to create a managed instance:

1. Select **Azure SQL** on the left menu of Azure portal. If **Azure SQL** is not in the list, select **All services**, and then enter **Azure SQL** in the search box.
2. Select **+Add** to open the **Select SQL deployment option** page. You can view additional information about an Azure SQL Database managed instance by selecting **Show details** on the **Managed instances** tile.
3. Select **Create**.



4. Use the tabs on the **Create Azure SQL Database Managed Instance** provisioning form to add required and optional information. The following sections describe these tabs.

### Basics

- Fill out mandatory information required on the **Basics** tab. This is a minimum set of information required to provision a managed instance.

Use the table below as a reference for information required at this tab.

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Subscription</b>	Your subscription.	A subscription that gives you permission to create new resources.
<b>Resource group</b>	A new or existing resource group.	For valid resource group names, see <a href="#">Naming rules and restrictions</a> .
<b>Managed instance name</b>	Any valid name.	For valid names, see <a href="#">Naming rules and restrictions</a> .
<b>Region</b>	The region in which you want to create the managed instance.	For information about regions, see <a href="#">Azure regions</a> .
<b>Managed instance admin login</b>	Any valid username.	For valid names, see <a href="#">Naming rules and restrictions</a> . Don't use "serveradmin" because that's a reserved server-level role.
<b>Password</b>	Any valid password.	The password must be at least 16 characters long and meet the <a href="#">defined complexity requirements</a> .

- Select **Configure Managed Instance** to size compute and storage resources and to review the pricing tiers. Use the sliders or text boxes to specify the amount of storage and the number of virtual cores. When you're finished, select **Apply** to save your selection.

**Configure performance**

**General Purpose**  
For most production workloads  
4 / 8 / 16 / 24 / 32 / 40 / 64 / 80 vCores  
32 GB - 8 TB storage capacity  
Fast storage

**Business Critical**  
For IO-intensive and compute-intensive workloads.  
4 / 8 / 16 / 24 / 32 / 40 / 64 / 80 vCores  
32 GB - 4 TB storage capacity  
Super fast storage

**Compute Generation**  
Hardware generation is managed and upgraded periodically. [Learn more](#)

**vCores** [What is a vCore?](#) 4

**Storage** in GB. Custom amount will be rounded to the nearest value divisible by 32. 256

**Days of backup retention** Gives you point-in-time restore capability within 7 days period.  
Enjoy your free trial. [Learn more](#)

**Save money** [?](#)  
Save up to 55% with a license you already own. Already have a SQL Server license?  
By selecting 'Yes', I confirm that I have a SQL Server license with Software Assurance to apply this [Azure Hybrid Benefit for SQL Server](#)

Monthly cost

**Apply**

- To review your choices before you create a managed instance, you can select **Review + create**. Or, configure networking options by selecting **Next: Networking**.

## Networking

- Fill out optional information on the **Networking** tab. If you omit this information, the portal will apply default settings.

**Create Azure SQL Database Managed Instance**

**Networking**

Configure virtual network and public endpoint connectivity for your Managed Instance. Define level of access and connection type. [Learn more](#)

**Virtual network**  
Select or create a virtual network to connect to your Managed Instance securely. Allow us to update subnet configuration for you automatically, or follow our guide to set it up yourself. [Learn more](#)

\* Virtual network [\(new\) vnet-my-managed-instance/ManagedInstance](#)  
New virtual network will be created. Network configuration required for Managed Instance will be applied to the subnet automatically.

**Connection type**  
Select a connection type to accelerate application access. This configuration will apply to virtual network and public endpoint. [Learn more](#)

Connection type (private endpoint) [Proxy \(Default\)](#)

**Public endpoint**  
Secure public endpoint provides the ability to connect to Managed Instance from the Internet without using VPN and is for data communication (TDS) only. Access is disabled by default unless explicitly allowed. [Learn more](#)

Public endpoint (data) [Disable](#) [Enable](#)

Accelerated networking is automatically enabled on Gen5 hardware. [Learn more](#)

**Review + create** < Previous Next : Additional settings >

Use the table below as a reference for information required at this tab.

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Virtual network</b>	Select either <b>Create new virtual network</b> or a valid virtual network and subnet.	If a network or subnet is unavailable, it must be <a href="#">modified to satisfy the network requirements</a> before you select it as a target for the new managed instance. For information about the requirements for configuring the network environment for a managed instance, see <a href="#">Configure a virtual network for a managed instance</a> .
<b>Connection type</b>	Choose between a proxy and a redirect connection type.	For more information about connection types, see <a href="#">Azure SQL Database connection policy</a> .
<b>Public endpoint</b>	Select <b>Enable</b> .	For a managed instance to be accessible through the public data endpoint, you need to enable this option.
<b>Allow access from (if Public endpoint is enabled)</b>	Select one of the options.	<p>The portal experience enables configuring a security group with a public endpoint.</p> <p>Based on your scenario, select one of the following options:</p> <ul style="list-style-type: none"> <li>• <b>Azure services:</b> We recommend this option when you're connecting from Power BI or another multitenant service.</li> <li>• <b>Internet:</b> Use for test purposes when you want to quickly spin up a managed instance. We don't recommend it for production environments.</li> <li>• <b>No access:</b> This option creates a <b>Deny</b> security rule. Modify this rule to make a managed instance accessible through a public endpoint.</li> </ul> <p>For more information on public endpoint security, see <a href="#">Using an Azure SQL Database managed instance securely with a public endpoint</a>.</p>

- Select **Review + create** to review your choices before you create a managed instance. Or, configure more custom settings by selecting **Next: Additional settings**.

## Additional settings

- Fill out optional information on the **Additional settings** tab. If you omit this information, the portal will apply default settings.

The screenshot shows the Microsoft Azure portal interface for creating a new Azure SQL Database Managed Instance. The top navigation bar includes the Microsoft Azure logo, a search bar, and user information (admin@contoso.com, contoso). The main title is 'Create Azure SQL Database Managed Instance'. Below the title, there are four tabs: 'Basics', 'Networking', 'Additional settings' (which is highlighted with a dashed border), and 'Review + create'. The 'Additional settings' section contains three main configuration groups: 'Collation', 'Time zone', and 'Geo-Replication'. Under 'Collation', it says 'Instance collation defines rules that sort and compare data, and cannot be changed after instance creation. The default instance collation is SQL\_Latin1\_CI\_AS. [Learn more](#)'. A dropdown menu shows 'SQL\_Latin1\_General\_CI\_AS' is selected. Under 'Time zone', it says 'Time zone is defined at the instance level and it applies to all databases created in this Managed Instance. Time zone cannot be changed after the instance creation. [Learn more](#)'. A dropdown menu shows '(UTC) Coordinated Universal Time' is selected. Under 'Geo-Replication', it says 'Use this instance as a Failover Group secondary. [Learn more](#)'. A radio button group shows 'Use as failover secondary' is set to 'No'.

Use the table below as a reference for information required at this tab.

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Collation</b>	Choose the collation that you want to use for your managed instance. If you migrate databases from SQL Server, check the source collation by using <pre>SELECT SERVERPROPERTY(N'Collation')</pre> and use that value.	For information about collations, see <a href="#">Set or change the server collation</a> .
<b>Time zone</b>	Select the time zone that your managed instance will observe.	For more information, see <a href="#">Time zones</a> .
<b>Use as failover secondary</b>	Select <b>Yes</b> .	Enable this option to use the managed instance as a failover group secondary.
<b>Primary managed instance</b> (if <b>Use as failover secondary</b> is set to <b>Yes</b> )	Choose an existing primary managed instance that will be joined in the same DNS zone with the managed instance you're creating.	This step will enable post-creation configuration of the failover group. For more information, see <a href="#">Tutorial: Add a SQL Database managed instance to a failover group</a> .

## Review + create

- Select **Review + create** tab to review your choices before you create the managed instance.

Microsoft Azure Search resources, services, and docs (G+) Dashboard > New > Azure SQL Managed Instance > Create Azure SQL Database Managed Instance admin@contoso.com CONTOSO

## Create Azure SQL Database Managed Instance

Microsoft

Deploying Managed Instance is a long running operation taking up to 6 hours to complete.

**Terms**

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. For additional details see [Azure Marketplace Terms](#).

**Basics**

Subscription	Dev/test subscription
Resource group	myRG
Managed Instance name	my-managed-instance
Region	(US) East US
Compute + storage	General Purpose: Gen5, 4 vCores, 32 GB storage
Managed Instance admin login	cloudsa

**Networking**

Virtual network	Create new virtual network
Prepare subnet for Managed Instance	Automatic
Connection type	Proxy
Public endpoint (data)	Disabled

**Additional settings**

Collation	SQL_Latin1_General_CI_AS
Time zone	(UTC) Coordinated Universal Time
Use as failover secondary	No

**Create** < Previous Next : Review + create > Download a template for automation

6. Select **Create** to start provisioning the managed instance.

### IMPORTANT

Deploying a managed instance is a long-running operation. Deployment of the first instance in the subnet typically takes much longer than deploying into a subnet with existing managed instances. For average provisioning times, see [Managed instance management operations](#).

### Monitor deployment progress

7. Select the **Notifications** icon to view the status of the deployment.

Notifications

More events in the activity log → Dismiss all

■■■ Deployment in progress... Running

Deployment to resource group 'myRG' is in progress.

7 minutes ago

8. Select **Deployment in progress** in the notification to open the managed instance window and further monitor the deployment progress.

## TIP

If you closed your web browser or moved away from the deployment progress screen, follow these steps to locate back the deployment progress screen:

1. In Azure portal, open the resource group (on the **Basics** tab) to which you're deploying a managed instance.
2. Select **Deployments**.
3. Select the managed instance deployment operation in progress.

## Post-deployment operations

To review resources created, fine-tune network settings, and retrieve host connection details (FQDN) follow steps described in this section.

### View resources created

Upon successful deployment of managed instance, to view resources created:

1. Open the resource group for your managed instance. View its resources that were created for you in the [Create a managed instance](#) quickstart.

NAME	TYPE	LOCATION
carlrab20190215mi	SQL managed instance	West US 2
nsg-carlrb20190215mi	Network security group	West US 2
rt-carlrb20190215mi	Route table	West US 2
VirtualClusterd8044b17-bbe9-4a3c-b456-b87016a928fb	Virtual cluster	West US 2
vnet-carlrb20190215mi	Virtual network	West US 2

### View and fine-tune network settings

To optionally fine-tune networking settings, inspect the following:

1. Select the route table to review the user-defined route (UDR) that was created for you.

NAME	TYPE	LOCATION
carlrab20190215mi	SQL managed instance	West US 2
nsg-carlrb20190215mi	Network security group	West US 2
rt-carlrb20190215mi	Route table	West US 2
VirtualClusterd8044b17-bbe9-4a3c-b456-b87016a928fb	Virtual cluster	West US 2
vnet-carlrb20190215mi	Virtual network	West US 2

2. In the route table, review the entries to route traffic from and within the managed instance's virtual network. If you create or configure your route table manually, ensure to create these entries in the managed instance route table.

Screenshot of the Microsoft Azure portal showing the 'Overview' page for a route table named 'rt-carlab20190215mi'. The 'Routes' section displays a list of routes with their names, address prefixes, next hops, and subnets. A mouse cursor is hovering over the 'Subnets' link in the left sidebar.

NAME	ADDRESS PREFIX	NEXT HOP
mi-0-5-nexthop-internet	0.0.0.0/5	Internet
mi-11-8-nexthop-internet	11.0.0.0/8	Internet
mi-12-6-nexthop-internet	12.0.0.0/6	Internet
mi-128-3-nexthop-internet	128.0.0.0/3	Internet
mi-16-4-nexthop-internet	16.0.0.0/4	Internet
mi-160-5-nexthop-internet	160.0.0.0/5	Internet
mi-168-6-nexthop-internet	168.0.0.0/6	Internet
mi-172-12-nexthop-internet	172.0.0.0/12	Internet
mi-172-128-9-nexthop-internet	172.128.0.0/9	Internet
mi-172-32-11-nexthop-internet	172.32.0.0/11	Internet
mi-172-64-10-nexthop-internet	172.64.0.0/10	Internet
mi-173-8-nexthop-internet	173.0.0.0/8	Internet
mi-174-7-nexthop-internet	174.0.0.0/7	Internet
mi-176-4-nexthop-internet	176.0.0.0/4	Internet
mi-192-128-11-nexthop-internet	192.128.0.0/11	Internet
mi-192-160-13-nexthop-internet	192.160.0.0/13	Internet

3. Return to the resource group, and select the network security group.

Screenshot of the Microsoft Azure portal showing the 'Overview' page for a resource group named 'carlabrg20190219'. The 'Resource group' icon is highlighted. The 'NSG' 'nsg-carlab20190215mi' is selected and highlighted with a red box. It is listed as a Network security group located in West US 2.

4. Review the inbound and outbound security rules.

The screenshot shows the Azure portal interface for managing a Network Security Group (NSG). The left sidebar lists various resource groups and settings. The main area displays two tables of security rules:

Inbound security rules	Priority	Name	Port	Protocol	Source	Destination	Action
	100	allow_management_inbound	9000,9003,1438,144...	TCP	Any	Any	Allow
	200	allow_missubnet_inbound	Any	Any	10.0.0.0/16	Any	Allow
	300	allow_health_probe_inbound	Any	Any	AzureLoadBalancer	Any	Allow
	1000	allow_tds_inbound	1433	TCP	VirtualNetwork	Any	Allow
	1100	allow_redirect_inbound	11000-11999	TCP	VirtualNetwork	Any	Allow
	1200	allow_geodr_inbound	5022	TCP	VirtualNetwork	Any	Allow
	4096	deny_all_inbound	Any	Any	Any	Any	Deny
	65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
	65001	AllowAzureLoadBalancerInBound	Any	Any	AzureLoadBalancer	Any	Allow
	65500	DenyAllInBound	Any	Any	Any	Any	Deny

Outbound security rules	Priority	Name	Port	Protocol	Source	Destination	Action
	100	allow_management_outbound	80,443,12000	TCP	Any	Internet	Allow
	200	allow_missubnet_outbound	Any	Any	Any	10.0.0.0/16	Allow
	1000	allow_linkedserver_outbound	1433	TCP	Any	VirtualNetwork	Allow
	1100	allow_redirect_outbound	11000-11999	TCP	Any	VirtualNetwork	Allow
	1200	allow_geodr_outbound	5022	TCP	Any	VirtualNetwork	Allow
	4096	deny_all_outbound	Any	Any	Any	Any	Deny
	65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
	65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
	65500	DenyAllOutBound	Any	Any	Any	Any	Deny

### IMPORTANT

If you have configured public endpoint for your managed instance, you need to open ports to allow network traffic allowing connections to managed instance from the public Internet, see [Configure a public endpoint for managed instance](#) for more information.

## Retrieve connection details to managed instance

To connect to managed instance, follow these steps to retrieve host name and fully qualified domain name (FQDN):

1. Return to the resource group and select your managed instance.

The screenshot shows the Azure portal interface for managing a resource group. The left sidebar lists various resource groups and settings. The main area displays a table of resources:

NAME	TYPE	LOCATION
carlrabrg20190219mi	SQL managed instance	West US 2
nsg-carlrb20190215mi	Network security group	West US 2
rt-carlrb20190215mi	Route table	West US 2
VirtualClusterd8044b17-bbe9-4a3c-b456-b87016a928fb	Virtual cluster	West US 2
vnet-carlrb20190215mi	Virtual network	West US 2

2. On the **Overview** tab, locate the **Host** property. Copy the host name for the managed instance for use in the next quickstart.

The screenshot shows the Azure portal interface for a SQL managed instance. The main title bar says "Microsoft Azure" and "Search resources, services, and docs". Below it, the breadcrumb navigation shows "Home > SQL managed instances > carlrab20190215mi". The left sidebar has a tree view with "Overview" selected (highlighted with a red box). The main content area displays various details about the managed instance, including its name, status, location, subscription, and tags. A prominent green message at the top right says "Your Managed Instance is ready. Click here to get started." To the right of the host name, there is a "Click to copy" button.

The value copied represents a fully qualified domain name (FQDN) that can be used to connect to managed instance. It is similar to the following address example:  
*your\_host\_name.a1b2c3d4e5f6.database.windows.net*.

## Next steps

To learn about how to connect to a managed instance:

- For an overview of the connection options for applications, see [Connect your applications to a managed instance](#).
- For a quickstart that shows how to connect to a managed instance from an Azure virtual machine, see [Configure an Azure virtual machine connection](#).
- For a quickstart that shows how to connect to a managed instance from an on-premises client computer by using a point-to-site connection, see [Configure a point-to-site connection](#).

To restore an existing SQL Server database from on-premises to a managed instance:

- Use the [Azure Database Migration Service for migration](#) to restore from a database backup file.
- Use the [T-SQL RESTORE command](#) to restore from a database backup file.

For advanced monitoring of managed instance database performance with built-in troubleshooting intelligence, see [Monitor Azure SQL Database by using Azure SQL Analytics](#).

# Use PowerShell to create an Azure SQL Database managed instance

11/6/2019 • 5 minutes to read • [Edit Online](#)

This PowerShell script example creates an Azure SQL Database managed instance in a dedicated subnet within a new virtual network. It also configures a route table and a network security group for the virtual network. Once the script has been successfully run, the managed instance can be accessed from within the virtual network or from an on-premises environment. See [Configure Azure VM to connect to an Azure SQL Database Managed Instance](#) and [Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises](#).

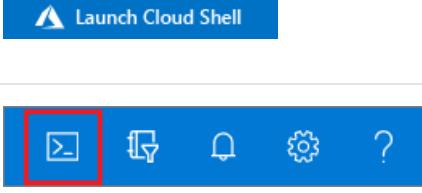
## IMPORTANT

For limitations, see [supported regions](#) and [supported subscription types](#).

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this tutorial requires AZ PowerShell 1.4.0 or later. If you need to upgrade, see [Install Azure PowerShell module](#). If you are running PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

## Sample script

```
Connect-AzAccount
The SubscriptionId in which to create these objects
$SubscriptionId = ''
Set the resource group name and location for your managed instance
$resourceGroupName = "myResourceGroup-$($Get-Random)"
.setLocation = "westus2"
Set the networking values for your managed instance
$vNetName = "myVnet-$($Get-Random)"
$vNetAddressPrefix = "10.0.0.0/16"
$defaultSubnetName = "myDefaultSubnet-$($Get-Random)"
$defaultSubnetAddressPrefix = "10.0.0.0/24"
$miSubnetName = "myMISubnet-$($Get-Random)"
$miSubnetAddressPrefix = "10.0.0.0/24"
#Set the managed instance name for the new managed instance
$instanceName = "myMIName-$($Get-Random)"
Set the admin login and password for your managed instance
$miAdminSqlLogin = "SqlAdmin"
$miAdminSqlPassword = "ChangeYourAdminPassword1"
Set the managed instance service tier, compute level, and license mode
$edition = "General Purpose"
$vCores = 8
$maxStorage = 256
$computeGeneration = "Gen4"
$license = "LicenseIncluded" #"$BasePrice" or LicenseIncluded if you have don't have SQL Server licence that can be used for AHB discount

Set subscription context
Set-AzContext -SubscriptionId $subscriptionId

Create a resource group
$resourceGroup = New-AzResourceGroup -Name $resourceGroupName -Location $location

Configure virtual network, subnets, network security group, and routing table
$virtualNetwork = New-AzVirtualNetwork `

 -ResourceGroupName $resourceGroupName `

 -Location $location `

 -Name $vNetName `

 -AddressPrefix $vNetAddressPrefix

 Add-AzVirtualNetworkSubnetConfig `

 -Name $miSubnetName `

 -VirtualNetwork $virtualNetwork `

 -AddressPrefix $miSubnetAddressPrefix `

 | Set-AzVirtualNetwork

$virtualNetwork = Get-AzVirtualNetwork -Name $vNetName -ResourceGroupName $resourceGroupName

$miSubnetConfig = Get-AzVirtualNetworkSubnetConfig `

 -Name $miSubnetName `

 -VirtualNetwork $virtualNetwork

$miSubnetConfigId = $miSubnetConfig.Id

$networkSecurityGroupMiManagementService = New-AzNetworkSecurityGroup `

 -Name 'myNetworkSecurityGroupMiManagementService' `

 -ResourceGroupName $resourceGroupName `

 -location $location

$routeTableMiManagementService = New-AzRouteTable `

 -Name 'myRouteTableMiManagementService' `

 -ResourceGroupName $resourceGroupName `

 -location $location

Set-AzVirtualNetworkSubnetConfig `

 -VirtualNetwork $virtualNetwork `

 -SubnetId $miSubnetConfigId `

 -RouteTable $routeTableMiManagementService
```

```

 -Name $miSubnetName
 -AddressPrefix $miSubnetAddressPrefix `
 -NetworkSecurityGroup $networkSecurityGroupMiManagementService `
 -RouteTable $routeTableMiManagementService | `
Set-AzVirtualNetwork

Get-AzNetworkSecurityGroup `
 -ResourceGroupName $resourceGroupName `
 -Name "myNetworkSecurityGroupMiManagementService" `
| Add-AzNetworkSecurityRuleConfig `
 -Priority 100 `
 -Name "allow_management_inbound" `
 -Access Allow `
 -Protocol Tcp `
 -Direction Inbound `
 -SourcePortRange * `
 -SourceAddressPrefix * `
 -DestinationPortRange 9000,9003,1438,1440,1452 `
 -DestinationAddressPrefix * `
| Add-AzNetworkSecurityRuleConfig `
 -Priority 200 `
 -Name "allow_misubnet_inbound" `
 -Access Allow `
 -Protocol * `
 -Direction Inbound `
 -SourcePortRange * `
 -SourceAddressPrefix $miSubnetAddressPrefix `
 -DestinationPortRange * `
 -DestinationAddressPrefix * `
| Add-AzNetworkSecurityRuleConfig `
 -Priority 300 `
 -Name "allow_health_probe_inbound" `
 -Access Allow `
 -Protocol * `
 -Direction Inbound `
 -SourcePortRange * `
 -SourceAddressPrefix AzureLoadBalancer `
 -DestinationPortRange * `
 -DestinationAddressPrefix * `
| Add-AzNetworkSecurityRuleConfig `
 -Priority 1000 `
 -Name "allow_tds_inbound" `
 -Access Allow `
 -Protocol Tcp `
 -Direction Inbound `
 -SourcePortRange * `
 -SourceAddressPrefix VirtualNetwork `
 -DestinationPortRange 1433 `
 -DestinationAddressPrefix * `
| Add-AzNetworkSecurityRuleConfig `
 -Priority 1100 `
 -Name "allow_redirect_inbound" `
 -Access Allow `
 -Protocol Tcp `
 -Direction Inbound `
 -SourcePortRange * `
 -SourceAddressPrefix VirtualNetwork `
 -DestinationPortRange 11000-11999 `
 -DestinationAddressPrefix * `
| Add-AzNetworkSecurityRuleConfig `
 -Priority 4096 `
 -Name "deny_all_inbound" `
 -Access Deny `
 -Protocol * `
 -Direction Inbound `
 -SourcePortRange * `
 -SourceAddressPrefix * `
 -DestinationPortRange * `
 -DestinationAddressPrefix * `

```

```

| Add-AzNetworkSecurityRuleConfig `
-Priority 100 `
-Name "allow_management_outbound" `
-Access Allow `
-Protocol Tcp `
-Direction Outbound `
-SourcePortRange * `
-SourceAddressPrefix * `
-DestinationPortRange 80,443,12000 `
-DestinationAddressPrefix * `
| Add-AzNetworkSecurityRuleConfig `
-Priority 200 `
-Name "allow_misubnet_outbound" `
-Access Allow `
-Protocol * `
-Direction Outbound `
-SourcePortRange * `
-SourceAddressPrefix * `
-DestinationPortRange * `
-DestinationAddressPrefix $miSubnetAddressPrefix `
| Add-AzNetworkSecurityRuleConfig `
-Priority 4096 `
-Name "deny_all_outbound" `
-Access Deny `
-Protocol * `
-Direction Outbound `
-SourcePortRange * `
-SourceAddressPrefix * `
-DestinationPortRange * `
-DestinationAddressPrefix * `
| Set-AzNetworkSecurityGroup

Get-AzRouteTable `
-ResourceGroupName $resourceGroupName `
-Name "myRouteTableMiManagementService" `
| Add-AzRouteConfig `
-Name "ToManagedInstanceManagementService" `
-AddressPrefix 0.0.0.0/0 `
-NextHopType Internet `
| Add-AzRouteConfig `
-Name "ToLocalClusterNode" `
-AddressPrefix $miSubnetAddressPrefix `
-NextHopType VnetLocal `
| Set-AzRouteTable

Create managed instance
New-AzSqlInstance -Name $instanceName `
-ResourceGroupName $resourceGroupName -Location westus2 -SubnetId $miSubnetConfigId `
-AdministratorCredential (Get-Credential) `
-StorageSizeInGB $maxStorage -VCore $vCores -Edition $edition `
-ComputeGeneration $computeGeneration -LicenseType $license

This script will take a minimum of 3 hours to create a new managed instance in a new virtual network.
A second managed instance is created much faster.

Clean up deployment
Remove-AzResourceGroup -ResourceGroupName $resourceGroupName

```

## Clean up deployment

Use the following command to remove the resource group and all resources associated with it.

```
Remove-AzResourceGroup -ResourceGroupName $resourcegroupName
```

# Script explanation

This script uses the following commands. Each command in the table links to command specific documentation.

COMMAND	NOTES
<a href="#">New-AzResourceGroup</a>	Creates a resource group in which all resources are stored.
<a href="#">New-AzVirtualNetwork</a>	Creates a virtual network
<a href="#">Add-AzVirtualNetworkSubnetConfig</a>	Adds a subnet configuration to a virtual network
<a href="#">Get-AzVirtualNetwork</a>	Gets a virtual network in a resource group
<a href="#">Set-AzVirtualNetwork</a>	Sets the goal state for a virtual network
<a href="#">Get-AzVirtualNetworkSubnetConfig</a>	Gets a subnet in a virtual network
<a href="#">Set-AzVirtualNetworkSubnetConfig</a>	Configures the goal state for a subnet configuration in a virtual network
<a href="#">New-AzRouteTable</a>	Creates a route table
<a href="#">Get-AzRouteTable</a>	Gets route tables
<a href="#">Set-AzRouteTable</a>	Sets the goal state for a route table
<a href="#">New-AzSqlInstance</a>	Creates an Azure SQL Database managed instance
<a href="#">Remove-AzResourceGroup</a>	Deletes a resource group including all nested resources.

## Next steps

For more information on the Azure PowerShell, see [Azure PowerShell documentation](#).

Additional SQL Database PowerShell script samples can be found in the [Azure SQL Database PowerShell scripts](#).

# Azure SQL Database instance pools (preview) how-to guide

1/8/2020 • 6 minutes to read • [Edit Online](#)

This article provides details on how to create and manage [instance pools](#).

## Instance pool operations

The following table shows the available operations related to instance pools and their availability in the Azure portal and PowerShell.

COMMAND	AZURE PORTAL	POWERSHELL
Create instance pool	No	Yes
Update instance pool (limited number of properties)	No	Yes
Check instance pool usage and properties	No	Yes
Delete instance pool	No	Yes
Create managed instance inside instance pool	No	Yes
Update managed instance resource usage	Yes	Yes
Check managed instance usage and properties	Yes	Yes
Delete managed instance from the pool	Yes	Yes
Create a database in managed instance placed in the pool	Yes	Yes
Delete a database from managed instance	Yes	Yes

### Available PowerShell commands

CMDLET	DESCRIPTION
<a href="#">New-AzSqlInstancePool</a>	Creates an Azure SQL Database instance pool.
<a href="#">Get-AzSqlInstancePool</a>	Returns information about Azure SQL instance pool.
<a href="#">Set-AzSqlInstancePool</a>	Sets properties for an Azure SQL Database instance pool.

CMDLET	DESCRIPTION
<a href="#">Remove-AzSqlInstancePool</a>	Removes an Azure SQL Database instance pool.
<a href="#">Get-AzSqlInstancePoolUsage</a>	Returns information about Azure SQL instance pool usage.

To use PowerShell, [install the latest version of PowerShell Core](#), and follow instructions to [Install the Azure PowerShell module](#).

For operations related to instances both inside pools and single instances, use the standard [managed instance commands](#), but the *instance pool name* property must be populated when using these commands for an instance in a pool.

## How to deploy managed instances into pools

The process of deploying an instance into a pool consists of the following two steps:

1. One-off instance pool deployment. This is a long running operation, where the duration is the same as deploying a [single instance created in an empty subnet](#).
2. Repetitive instance deployment in an instance pool. The instance pool parameter must be explicitly specified as part of this operation. This is a relatively fast operation that typically takes up to 5 minutes.

In public preview, both steps are only supported using PowerShell and Resource Manager templates. The Azure portal experience is not currently available.

After a managed instance is deployed to a pool, you *can* use the Azure portal to change its properties on the pricing tier page.

## Create an instance pool

To create an instance pool:

1. [Create a virtual network with a subnet](#).
2. [Create an instance pool](#).

### Create a virtual network with a subnet

To place multiple instance pools inside the same virtual network, see the following articles:

- [Determine VNet subnet size for an Azure SQL Database managed instance](#).
- Create new virtual network and subnet using the [Azure portal template](#) or follow the instructions for [preparing an existing virtual network](#).

### Create an instance pool

After completing the previous steps, you are ready to create an instance pool.

The following restrictions apply to instance pools:

- Only General Purpose and Gen5 are available in public preview.
- Pool name can contain only lowercase, numbers and hyphen, and can't start with a hyphen.
- If you want to use AHB (Azure Hybrid Benefit), it is applied at the instance pool level. You can set the license type during pool creation or update it anytime after creation.

## IMPORTANT

Deploying an instance pool is a long running operation that takes approximately 4.5 hours.

To get network parameters:

```
$virtualNetwork = Get-AzVirtualNetwork -Name "miPoolVirtualNetwork" -ResourceGroupName "myResourceGroup"
$subnet = Get-AzVirtualNetworkSubnetConfig -Name "miPoolSubnet" -VirtualNetwork $virtualNetwork
```

To create an instance pool:

```
$instancePool = New-AzSqlInstancePool `
 -ResourceGroupName "myResourceGroup" `
 -Name "mi-pool-name" `
 -SubnetId $subnet.Id `
 -LicenseType "LicenseIncluded" `
 -VCore 80 `
 -Edition "GeneralPurpose" `
 -ComputeGeneration "Gen5" `
 -Location "westeurope"
```

## IMPORTANT

Because deploying an instance pool is a long running operation, you need to wait until it completes before running any of the following steps in this article.

## Create a managed instance inside the pool

After the successful deployment of the instance pool, it's time to create an instance inside it.

To create a managed instance, execute the following command:

```
$instanceOne = $instancePool | New-AzSqlInstance -Name "mi-pool-name" -VCore 2 -StorageSizeInGB 256
```

Deploying an instance inside a pool takes a couple of minutes. After the first instance has been created, additional instances can be created:

```
$instanceTwo = $instancePool | New-AzSqlInstance -Name "mi-pool-name" -VCore 4 -StorageSizeInGB 512
```

## Create a database inside an instance

To create and manage databases in a managed instance that's inside a pool, use the single instance commands.

To create a database inside a managed instance:

```
$poolinstancedb = New-AzSqlInstanceDatabase -Name "mipooldb1" -InstanceName "poolmi-001" -ResourceGroupName
"myResourceGroup"
```

## Get instance pool usage

To get a list of instances inside a pool:

```
$instancePool | Get-AzSqlInstance
```

To get pool resource usage:

```
$instancePool | Get-AzSqlInstancePoolUsage
```

To get detailed usage overview of the pool and instances inside it:

```
$instancePool | Get-AzSqlInstancePoolUsage -ExpandChildren
```

To list the databases in an instance:

```
$databases = Get-AzSqlInstanceDatabase -InstanceName "pool-mi-001" -ResourceGroupName "resource-group-name"
```

**NOTE**

There is a limit of 100 databases per pool (not per instance).

## Scale a managed instance inside a pool

After populating a managed instance with databases, you may hit instance limits regarding storage or performance. In that case, if pool usage has not been exceeded, you can scale your instance. Scaling a managed instance inside a pool is an operation that takes a couple of minutes. The prerequisite for scaling is available vCores and storage on the instance pool level.

To update the number of vCores and storage size:

```
$instanceOne | Set-AzSqlInstance -VCore 8 -StorageSizeInGB 512 -InstancePoolName "mi-pool-name"
```

To update storage size only:

```
$instance | Set-AzSqlInstance -StorageSizeInGB 1024 -InstancePoolName "mi-pool-name"
```

## Connect to a managed instance inside a pool

To connect to a managed instance in a pool, the following two steps are required:

1. [Enable the public endpoint for the instance](#).
2. [Add an inbound rule to the network security group \(NSG\)](#).

After both steps are complete, you can connect to the instance by using a public endpoint address, port, and credentials provided during instance creation.

### Enable the public endpoint for the instance

Enabling the public endpoint for an instance can be done through the Azure portal or by using the following PowerShell command:

```
$instanceOne | Set-AzSqlInstance -InstancePoolName "pool-mi-001" -PublicDataEndpointEnabled $true
```

This parameter can be set during instance creation as well.

### Add an inbound rule to the network security group

This step can be done through the Azure portal or using PowerShell commands, and can be done anytime after the subnet is prepared for the managed instance.

For details, see [Allow public endpoint traffic on the network security group](#).

## Move an existing single instance inside an instance pool

Moving instances in and out of a pool is one of the public preview limitations. A workaround that can be used relies on point-in-time restore of databases from an instance outside a pool to an instance that's already in a pool.

Both instances must be in the same subscription and region. Cross-region and cross-subscription restore is not currently supported.

This process does have a period of downtime.

To move existing databases:

1. Pause workloads on the managed instance you are migrating from.
2. Generate scripts to create system databases and execute them on the instance that's inside the instance pool.
3. Do a point-in-time restore of each database from the single instance to the instance in the pool.

```
$resourceGroupName = "my resource group name"
$managedInstanceName = "my managed instance name"
$databaseName = "my source database name"
$pointInTime = "2019-08-21T08:51:39.3882806Z"
$targetDatabase = "name of the new database that will be created"
$targetResourceGroupName = "resource group of instance pool"
$targetInstanceName = "pool instance name"

Restore-AzSqlInstanceDatabase -FromPointInTimeBackup `
 -ResourceGroupName $resourceGroupName `
 -InstanceName $managedInstanceName `
 -Name $databaseName `
 -PointInTime $pointInTime `
 -TargetInstanceDatabaseName $targetDatabase `
 -TargetResourceGroupName $targetResourceGroupName `
 -TargetInstanceName $targetInstanceName
```

4. Point your application to the new instance and resume its workloads.

If there are multiple databases, repeat the process for each database.

## Next steps

- For a features and comparison list, see [SQL common features](#).
- For more information about VNet configuration, see [managed instance VNet configuration](#).
- For a quickstart that creates a managed instance and restores a database from a backup file, see [create a managed instance](#).
- For a tutorial using the Azure Database Migration Service (DMS) for migration, see [managed instance migration using DMS](#).
- For advanced monitoring of managed instance database performance with built-in troubleshooting intelligence, see [Monitor Azure SQL Database using Azure SQL Analytics](#).
- For pricing information, see [SQL Database managed instance pricing](#).



# Configure public endpoint in Azure SQL Database managed instance

11/20/2019 • 3 minutes to read • [Edit Online](#)

Public endpoint for a [managed instance](#) enables data access to your managed instance from outside the [virtual network](#). You are able to access your managed instance from multi-tenant Azure services like Power BI, Azure App Service, or an on-premises network. By using the public endpoint on a managed instance, you do not need to use a VPN, which can help avoid VPN throughput issues.

In this article, you'll learn how to:

- Enable public endpoint for your managed instance in the Azure portal
- Enable public endpoint for your managed instance using PowerShell
- Configure your managed instance network security group to allow traffic to the managed instance public endpoint
- Obtain the managed instance public endpoint connection string

## Permissions

Due to the sensitivity of data that is in a managed instance, the configuration to enable managed instance public endpoint requires a two-step process. This security measure adheres to separation of duties (SoD):

- Enabling public endpoint on a managed instance needs to be done by the managed instance admin. The managed instance admin can be found on **Overview** page of your SQL managed instance resource.
- Allowing traffic using a network security group that needs to be done by a network admin. For more information, see [network security group permissions](#).

## Enabling public endpoint for a managed instance in the Azure portal

1. Launch the Azure portal at <https://portal.azure.com/>.
2. Open the resource group with the managed instance, and select the **SQL managed instance** that you want to configure public endpoint on.
3. On the **Security** settings, select the **Virtual network** tab.
4. In the Virtual network configuration page, select **Enable** and then the **Save** icon to update the configuration.

The screenshot shows the Azure portal interface for managing a SQL managed instance named 'myfirstmanagedinstance'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with Quick start, Connection strings, Active Directory admin, Pricing tier, Instance Failover Groups, Properties, Locks, Export template), and Security (with Advanced Data Security, Virtual network selected, and Transparent data encryption). The main content area displays the 'Virtual network' settings, including a warning about enabling public endpoints, the host name (myfirstmanagedinstance.public.24e0bf8017f2.database.windows.net), port (3342), and connection type (Proxy (Default)). A red box highlights the 'Enable' button in the 'Public endpoint' section.

## Enabling public endpoint for a managed instance using PowerShell

### Enable public endpoint

Run the following PowerShell commands. Replace **subscription-id** with your subscription ID. Also replace **rg-name** with the resource group for your managed instance, and replace **mi-name** with the name of your managed instance.

```
Install-Module -Name Az

Import-Module Az.Accounts
Import-Module Az.Sql

Connect-AzAccount

Use your subscription ID in place of subscription-id below

Select-AzSubscription -SubscriptionId {subscription-id}

Replace rg-name with the resource group for your managed instance, and replace mi-name with the name of your managed instance

$mi = Get-AzSqlInstance -ResourceGroupName {rg-name} -Name {mi-name}

$mi = $mi | Set-AzSqlInstance -PublicDataEndpointEnabled $true -force
```

### Disable public endpoint

To disable the public endpoint using PowerShell, you would execute the following command (and also do not

forget to close the NSG for the inbound port 3342 if you have it configured):

```
Set-AzSqlInstance -PublicDataEndpointEnabled $false -force
```

## Allow public endpoint traffic on the network security group

1. If you have the configuration page of the managed instance still open, navigate to the **Overview** tab. Otherwise, go back to your **SQL managed instance** resource. Select the **Virtual network/subnet** link, which will take you to the Virtual network configuration page.

The screenshot shows the Azure portal's Overview page for a managed instance named 'MIEPTest'. In the 'Virtual network/subnet' section, the value 'vnet-myfirstmanagedinstance/Managedinstance' is highlighted with a red box. Other details shown include the managed instance admin ('miadmin'), host ('myfirstmanagedinstance.24e0bf8017f2.database.windows.net'), pricing tier ('General Purpose Gen5 (256 GB, 8 vCores)'), and virtual cluster ('VirtualCluster2bbc3bd-2307-4127-8b60-b445bc0e1603').

2. Select the **Subnets** tab on the left configuration pane of your Virtual network, and make note of the **SECURITY GROUP** for your managed instance.

The screenshot shows the 'Subnets' page for a virtual network named 'vnet-myfirstmanagedinstance'. A subnet named 'Managedinstance' is selected, and its 'SECURITY GROUP' field, which contains the value 'nsg-myfirstmanagedinstance', is highlighted with a red box.

3. Go back to your resource group that contains your managed instance. You should see the **Network security group** name noted above. Select the name to go into the network security group configuration page.

4. Select the **Inbound security rules** tab, and **Add** a rule that has higher priority than the **deny\_all\_inbound** rule with the following settings:

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Source</b>	Any IP address or Service tag	<ul style="list-style-type: none"><li>For Azure services like Power BI, select the Azure Cloud Service Tag</li><li>For your computer or Azure VM, use NAT IP address</li></ul>
<b>Source port ranges</b>	*	Leave this to * (any) as source ports are usually dynamically allocated and as such, unpredictable
<b>Destination</b>	Any	Leaving destination as Any to allow traffic into the managed instance subnet
<b>Destination port ranges</b>	3342	Scope destination port to 3342, which is the managed instance public TDS endpoint

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Protocol</b>	TCP	Managed instance uses TCP protocol for TDS
<b>Action</b>	Allow	Allow inbound traffic to managed instance through the public endpoint
<b>Priority</b>	1300	Make sure this rule is higher priority than the <b>deny_all_inbound</b> rule

1100	allow_redirect_inbound	11000-11999	TCP	VirtualNetwork	Any	<input checked="" type="radio"/> Allow	...
1200	allow_geodr_inbound	5022	TCP	VirtualNetwork	Any	<input checked="" type="radio"/> Allow	...
1300	public_endpoint_inbound	3342	TCP	Any	Any	<input checked="" type="radio"/> Allow	...
4096	deny_all_inbound	Any	Any	Any	Any	<input type="radio"/> Deny	...

#### NOTE

Port 3342 is used for public endpoint connections to managed instance, and cannot be changed at this point.

## Obtaining the managed instance public endpoint connection string

1. Navigate to the SQL managed instance configuration page that has been enabled for public endpoint. Select the **Connection strings** tab under the **Settings** configuration.
2. Note that the public endpoint host name comes in the format **<mi\_name>.public.<dns\_zone>.database.windows.net** and that the port used for the connection is 3342.

## Next steps

- Learn about [using Azure SQL Database managed instance securely with public endpoint](#).

# Quickstart: Configure Azure VM to connect to an Azure SQL Database Managed Instance

11/8/2019 • 5 minutes to read • [Edit Online](#)

This quickstart shows you how to configure an Azure virtual machine to connect to an Azure SQL Database Managed Instance using SQL Server Management Studio (SSMS). For a quickstart showing how to connect from an on-premises client computer using a point-to-site connection, see [Configure a point-to-site connection](#)

## Prerequisites

This quickstart uses the resources created in [Create a Managed Instance](#) as its starting point.

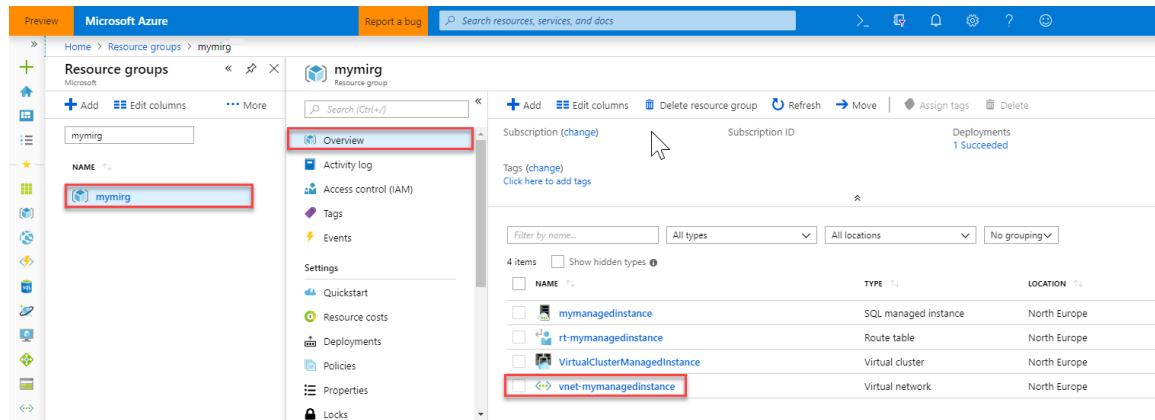
## Sign in to the Azure portal

Sign in to the [Azure portal](#).

## Create a new subnet in the Managed Instance VNet

The following steps create a new subnet in the Managed Instance VNet so an Azure virtual machine can connect to the Managed Instance. The Managed Instance subnet is dedicated to Managed Instances. You can't create any other resources, like Azure virtual machines, in that subnet.

1. Open the resource group for the Managed Instance that you created in the [Create a Managed Instance](#) quickstart. Select the virtual network for your Managed Instance.



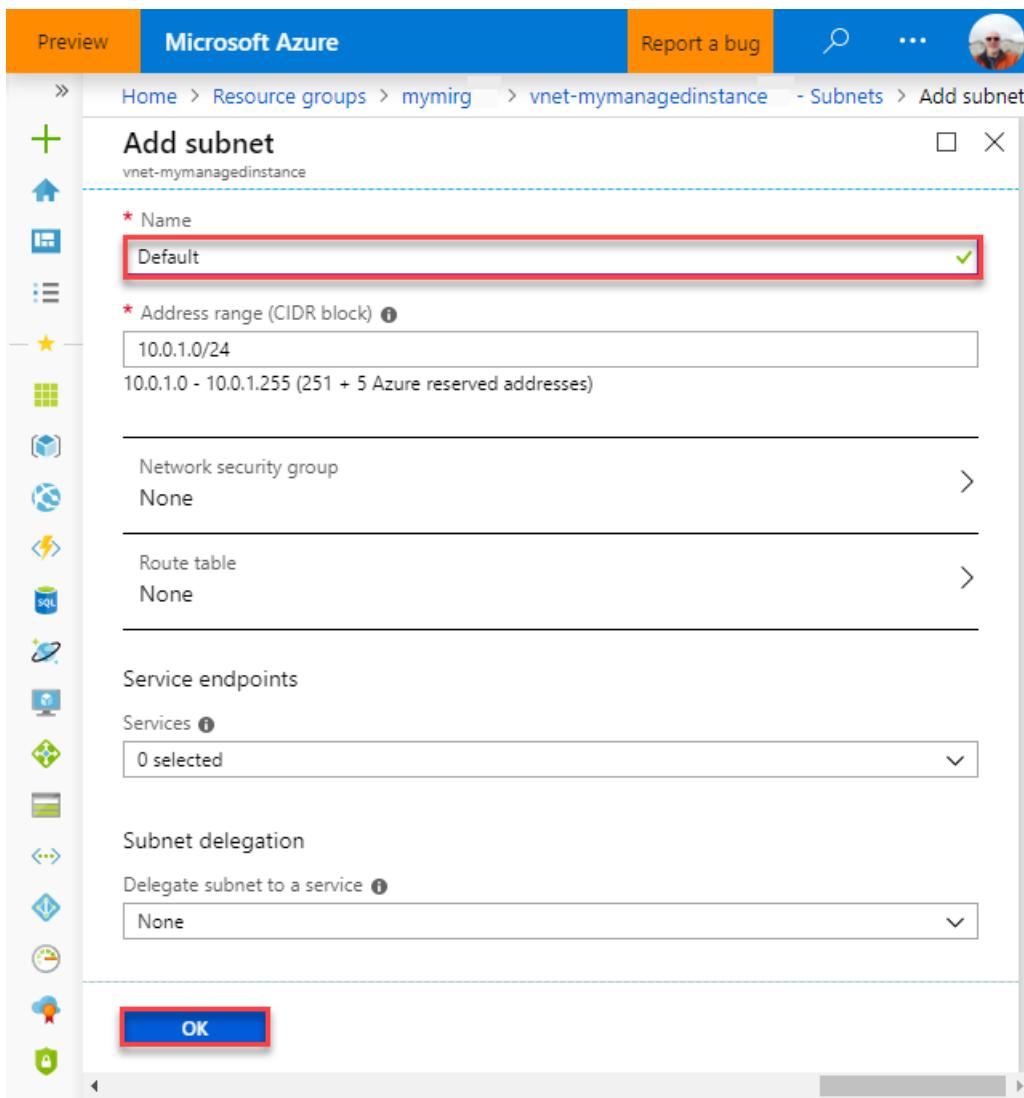
NAME	TYPE	LOCATION
mymanagedinstance	SQL managed instance	North Europe
rt-mymanagedinstance	Route table	North Europe
virtualClusterManagedInstance	Virtual cluster	North Europe
vnet-mymanagedinstance	Virtual network	North Europe

2. Select **Subnets** and then select **+ Subnet** to create a new subnet.

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Preview', 'Microsoft Azure', 'Report a bug', and various icons. Below the navigation bar, the URL path is visible: Home > Resource groups > mymigr > vnet-mymanagedinstance - Subnets. On the left, a sidebar lists several options like Overview, Activity log, Tags, etc., with 'Subnets' being the active section, also highlighted with a red box. The main content area shows a table of subnets. A red box highlights the '+ Subnet' button in the top right corner of the table header. The table has columns for NAME, ADDRESS RANGE, AVAILABLE ADDR..., and SECURITY GROUP. One row is shown: ManagedInstance, 10.0.0.0/24, 251, and a blank security group field.

3. Fill out the form using the information in this table:

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Name</b>	Any valid name	For valid names, see <a href="#">Naming rules and restrictions</a> .
<b>Address range (CIDR block)</b>	A valid range	The default value is good for this quickstart.
<b>Network security group</b>	None	The default value is good for this quickstart.
<b>Route table</b>	None	The default value is good for this quickstart.
<b>Service endpoints</b>	0 selected	The default value is good for this quickstart.
<b>Subnet delegation</b>	None	The default value is good for this quickstart.



SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Subscription</b>	A valid subscription	Must be a subscription in which you have permission to create new resources.
<b>Resource Group</b>	The resource group that you specified in the <a href="#">Create Managed Instance</a> quickstart.	This resource group must be the one in which the VNet exists.
<b>Location</b>	The location for the resource group	This value is populated based on the resource group selected.
<b>Virtual machine name</b>	Any valid name	For valid names, see <a href="#">Naming rules and restrictions</a> .
<b>Admin Username</b>	Any valid username	For valid names, see <a href="#">Naming rules and restrictions</a> . Don't use "serveradmin" as that is a reserved server-level role. You use this username any time you <a href="#">connect to the VM</a> .
<b>Password</b>	Any valid password	The password must be at least 12 characters long and meet the <a href="#">defined complexity requirements</a> . You use this password any time you <a href="#">connect to the VM</a> .
<b>Virtual Machine Size</b>	Any valid size	The default in this template of <b>Standard_B2s</b> is sufficient for this quickstart.
<b>Location</b>	[resourceGroup().location].	Don't change this value.
<b>Virtual Network Name</b>	The virtual network in which you created the Managed Instance.	
<b>Subnet name</b>	The name of the subnet that you created in the previous procedure	Don't choose the subnet in which you created the Managed Instance.
<b>artifacts Location</b>	[deployment().properties.templateLink.uri]	Don't change this value.
<b>artifacts Location Sas token</b>	leave blank	Don't change this value.

**Custom deployment**  
Deploy from a custom template

**TEMPLATE**

201-vm-win-vnet-sql-tools  
4 resources

**BASICS**

- \* Subscription: mymirg
- \* Resource group: mymirg
- \* Location: North Europe

**SETTINGS**

- \* Virtual Machine Name: mymivmq
- \* Admin Username: mymivmqadmin
- \* Admin Password: [REDACTED]
- Virtual Machine Size: Standard\_B2s
- Location: [resourceGroup().location]
- Virtual Network Name: vnet-mymanagedinstance
- Subnet Name: Default
- Artifacts Location: [deployment().properties.templateLink.uri]
- Artifacts Location Sas Token: [REDACTED]

**TERMS AND CONDITIONS**

Template information | Azure Marketplace Terms | Azure Marketplace

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated with the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

I agree to the terms and conditions stated above

**Purchase**

If you used the suggested VNet name and the default subnet in [creating your Managed Instance](#), you don't need to change last two parameters. Otherwise you should change these values to the values that you entered when you set up the network environment.

3. Select the **I agree to the terms and conditions stated above** checkbox.
4. Select **Purchase** to deploy the Azure VM in your network.
5. Select the **Notifications** icon to view the status of deployment.

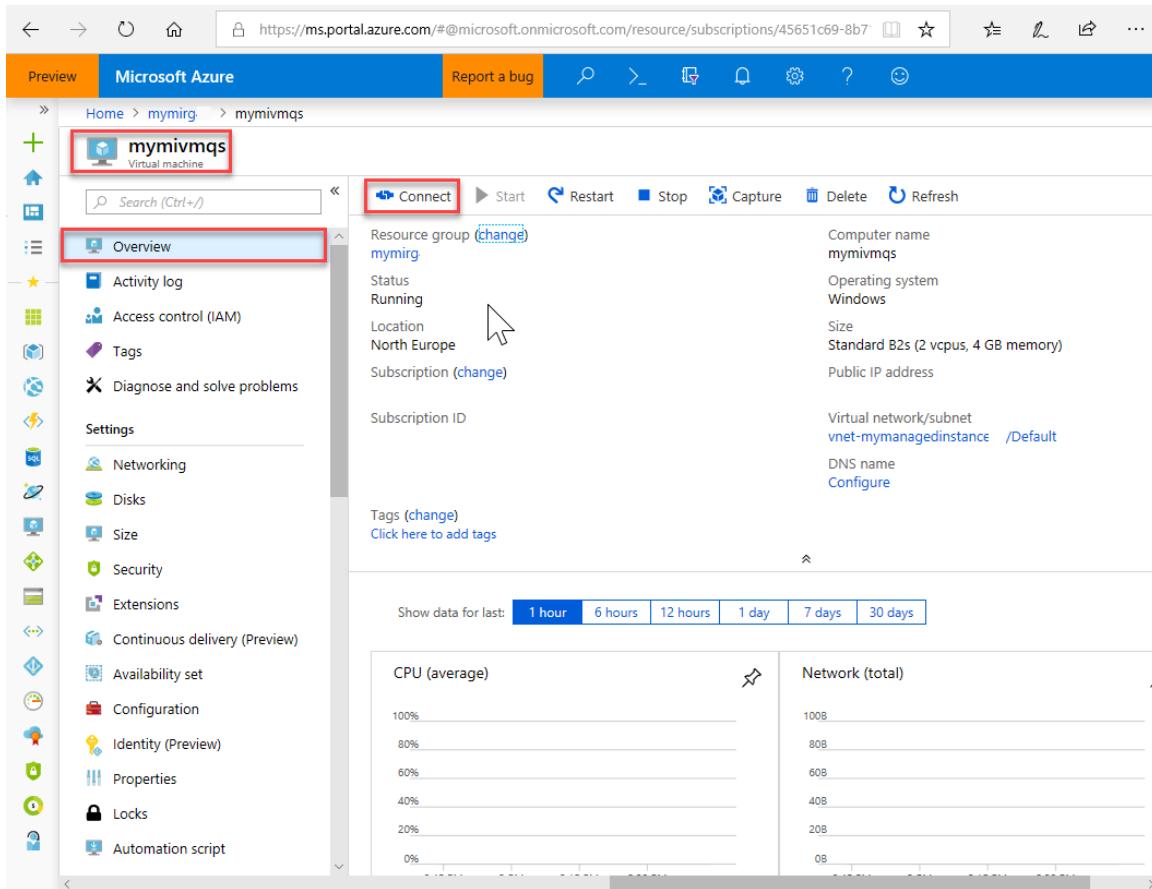
## IMPORTANT

Do not continue until approximately 15 minutes after the virtual machine is created to give time for the post-creation scripts to install SQL Server Management Studio.

## Connect to virtual machine

The following steps show you how to connect to your newly created virtual machine using a remote desktop connection.

1. After deployment completes, go to the virtual machine resource.



The screenshot shows the Microsoft Azure portal interface. In the center, the 'Overview' page for a virtual machine named 'mymivmq' is displayed. The 'Connect' button, located at the top right of the main content area, is highlighted with a red box. On the left, a navigation menu lists various options like Activity log, Access control (IAM), Tags, and Settings. The 'Overview' section contains details such as Resource group (mymirg), Status (Running), Location (North Europe), Subscription (change), and various connectivity and monitoring metrics like CPU (average) and Network (total).

2. Select **Connect**.

A Remote Desktop Protocol file (.rdp file) form appears with the public IP address and port number for the virtual machine.

Connect to virtual machine X  
quickstartmivm

RDP SSH

To connect to your virtual machine via RDP, select an IP address, optionally change the port number, and download the RDP file.

\* IP address

Public IP address (40.115.106.63) ▼

\* Port number

3389

**Download RDP File**



Inbound traffic to the Public IP address may be blocked.  
You can update inbound port rules in the **VM Networking** page.



You can troubleshoot VM connection issues by opening  
the **Diagnose and solve problems** page.

3. Select **Download RDP File**.

**NOTE**

You can also use SSH to connect to your VM.

4. Close the **Connect to virtual machine** form.
5. To connect to your VM, open the downloaded RDP file.
6. When prompted, select **Connect**. On a Mac, you need an RDP client such as this [Remote Desktop Client](#) from the Mac App Store.
7. Enter the username and password you specified when creating the virtual machine, then choose **OK**.
8. You might receive a certificate warning during the sign-in process. Choose **Yes** or **Continue** to proceed with the connection.

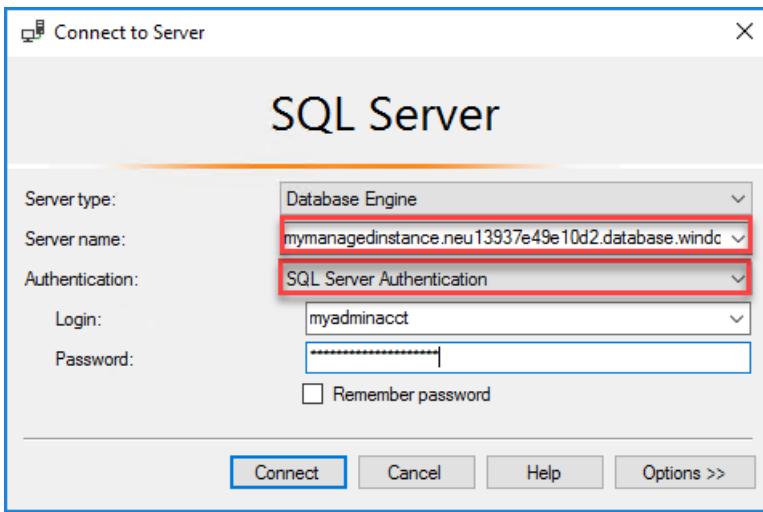
You're connected to your virtual machine in the Server Manager dashboard.

## Use SSMS to connect to the Managed Instance

1. In the virtual machine, open SQL Server Management Studio (SSMS).

It takes a few moments to open as it needs to complete its configuration since this is the first time SSMS has been started.

2. In the **Connect to Server** dialog box, enter the fully qualified **host name** for your Managed Instance in the **Server name** box. Select **SQL Server Authentication**, provide your username and password, and then select **Connect**.



After you connect, you can view your system and user databases in the Databases node, and various objects in the Security, Server Objects, Replication, Management, SQL Server Agent, and XEvent Profiler nodes.

## Next steps

- For a quickstart showing how to connect from an on-premises client computer using a point-to-site connection, see [Configure a point-to-site connection](#).
- For an overview of the connection options for applications, see [Connect your applications to Managed Instance](#).
- To restore an existing SQL Server database from on-premises to a Managed instance, you can use the [Azure Database Migration Service \(DMS\) for migration](#) or the [T-SQL RESTORE command](#) to restore from a database backup file.

# Quickstart: Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises

11/7/2019 • 3 minutes to read • [Edit Online](#)

This quickstart demonstrates how to connect to an Azure SQL Database Managed Instance using [SQL Server Management Studio \(SSMS\)](#) from an on-premises client computer over a point-to-site connection. For information about point-to-site connections, see [About Point-to-Site VPN](#)

## Prerequisites

This quickstart:

- Uses the resources created [Create a Managed Instance](#) as its starting point.
- Requires PowerShell 5.1 and AZ PowerShell 1.4.0 or later on your on-premises client computer. If necessary, see the instructions for [installing the Azure PowerShell module](#).
- Requires the newest version of [SQL Server Management Studio \(SSMS\)](#) on your on-premises client computer.

## Attach a VPN gateway to your Managed Instance virtual network

1. Open PowerShell on your on-premises client computer.
2. Copy this PowerShell script. This script attaches a VPN Gateway to the Managed Instance virtual network that you created in the [Create a Managed Instance](#) quickstart. This script uses the Azure PowerShell Az Module and will do the following for either Windows or Linux based hosts:
  - Creates and install certificates on client machine
  - Calculates the future VPN Gateway subnet IP range
  - Creates the GatewaySubnet
  - Deploys the Azure Resource Manager template that attaches the VPN Gateway to VPN subnet

```
$scriptUrlBase = 'https://raw.githubusercontent.com/Microsoft/sql-server-samples/master/samples/manage/azure-sql-db-managed-instance/attach-vpn-gateway'

$parameters = @{
 subscriptionId = '<subscriptionId>'
 resourceGroupName = '<resourceGroupName>'
 virtualNetworkName = '<virtualNetworkName>'
 certificateNamePrefix = '<certificateNamePrefix>'
}

Invoke-Command -ScriptBlock ([Scriptblock]::Create((iwr ($scriptUrlBase+'/attachVPNGateway.ps1?t=' + [DateTime]::Now.Ticks)).Content)) -ArgumentList $parameters, $scriptUrlBase
```

3. Paste the script in your PowerShell window and provide the required parameters. The values for `<subscriptionId>`, `<resourceGroup>`, and `<virtualNetworkName>` should match the ones that you used for the [Create Managed Instance](#) quickstart. The value for `<certificateNamePrefix>` can be a string of

your choice.

4. Execute the PowerShell script.

**IMPORTANT**

Do not continue until the PowerShell script completes.

## Create a VPN connection to your Managed Instance

1. Sign in to the [Azure portal](#).
2. Open the resource group in which you created the virtual network gateway, and then open the virtual network gateway resource.
3. Select **Point-to-site configuration** and then select **Download VPN client**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'Point-to-site configuration' item selected, indicated by a red box. The main content area shows a 'Gateway-rlbnu2na36uxo - Point-to-site configuration' page. A 'Download VPN client' button is highlighted with a red box. Other visible sections include 'Address pool' (192.168.0.0/24), 'Root certificates', and 'Revoked certificates'.

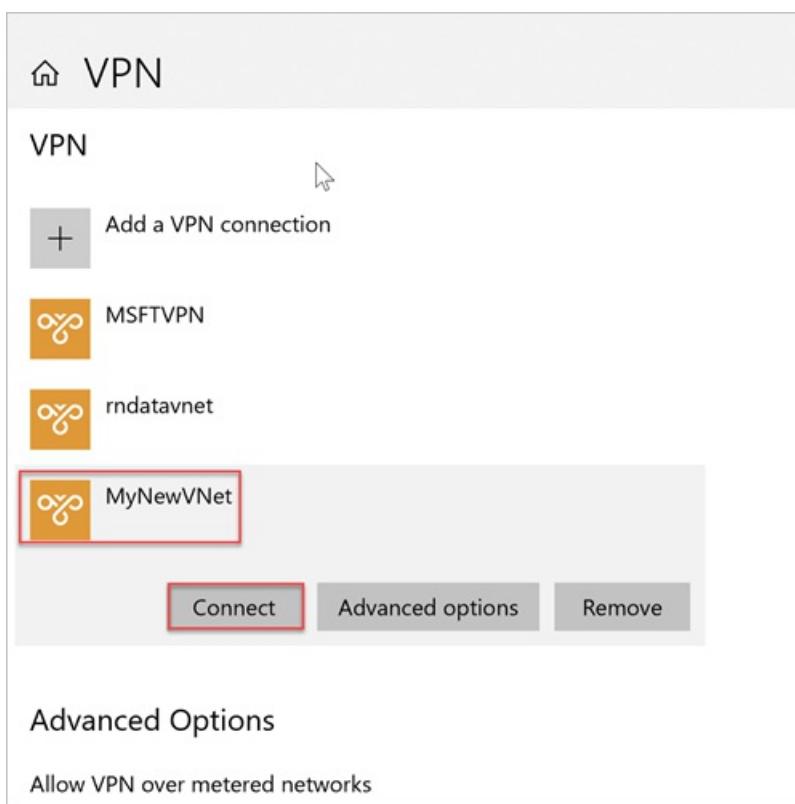
4. On your on-premises client computer, extract the files from the zip file and then open the folder with the extracted files.
5. Open the `WindowsAmd64` folder and open the **VpnClientSetupAmd64.exe** file.
6. If you receive a **Windows protected your PC** message, click **More info** and then click **Run anyway**.



7. In the User Account Control dialog box, click **Yes** to continue.
8. In the dialog box referencing your virtual network, select **Yes** to install the VPN Client for your virtual network.

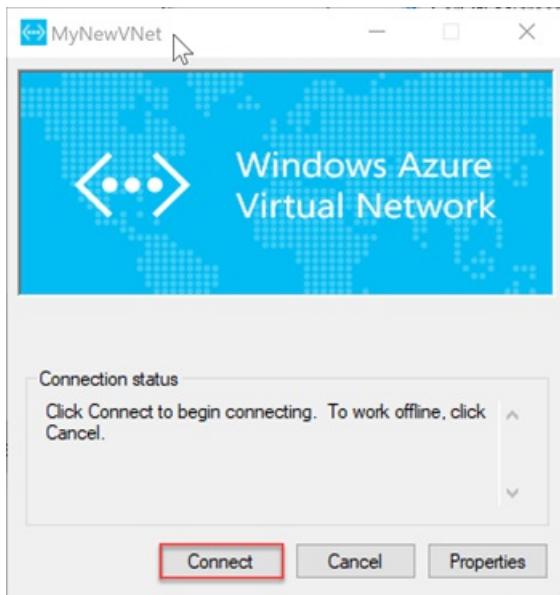
## Connect to the VPN connection

1. Go to **VPN** in **Network & Internet** on your on-premises client computer and select your Managed Instance virtual network to establish a connection to this VNet. In the following image, the VNet is named **MyNewVNet**.



2. Select **Connect**.

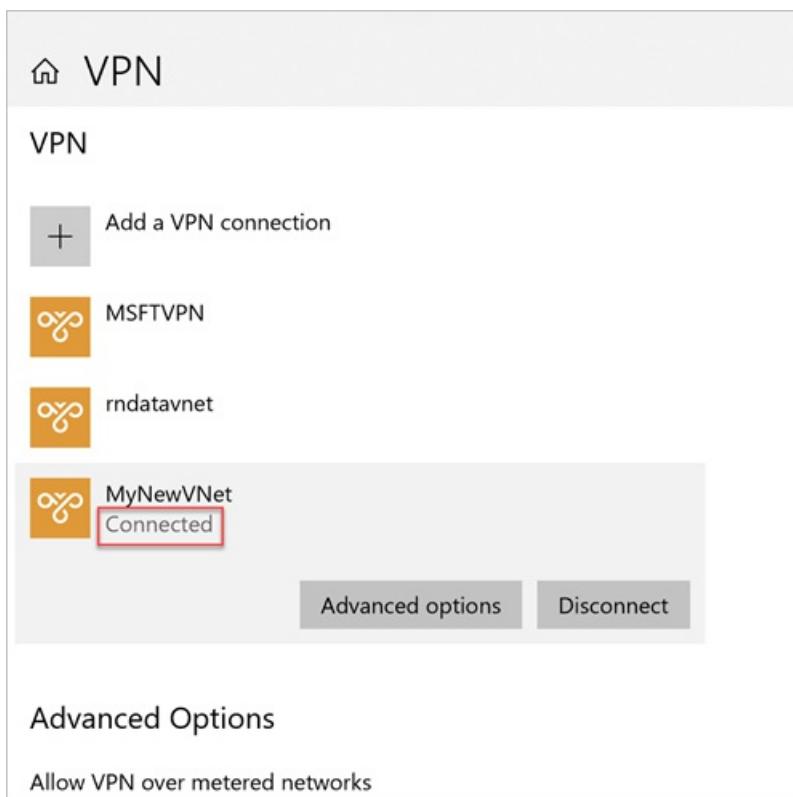
3. In the dialog box, select **Connect**.



4. When prompted that Connection Manager needs elevated privilege to update your route table, choose **Continue**.

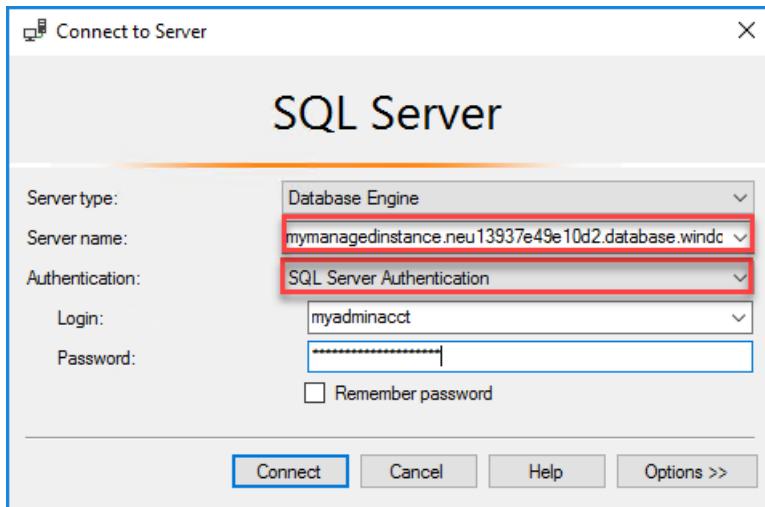
5. Select **Yes** in the User Account Control dialog box to continue.

You've established a VPN connection to your Managed Instance VNet.



## Use SSMS to connect to the Managed Instance

1. On the on-premises client computer, open SQL Server Management Studio (SSMS).
2. In the **Connect to Server** dialog box, enter the fully qualified **host name** for your Managed Instance in the **Server name** box.
3. Select **SQL Server Authentication**, provide your username and password, and then select **Connect**.



After you connect, you can view your system and user databases in the Databases node. You can also view various objects in the Security, Server Objects, Replication, Management, SQL Server Agent, and XEvent Profiler nodes.

## Next steps

- For a quickstart showing how to connect from an Azure virtual machine, see [Configure a point-to-site connection](#).
- For an overview of the connection options for applications, see [Connect your applications to Managed Instance](#).
- To restore an existing SQL Server database from on-premises to a Managed Instance, you can use the [Azure Database Migration Service \(DMS\) for migration](#) or the [T-SQL RESTORE command](#) to restore from a database backup file.

# Quickstart: Restore a database to a Managed Instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

In this quickstart, you'll use SQL Server Management Studio (SSMS) to restore a database (the Wide World Importers - Standard backup file) from Azure Blob storage into an Azure SQL Database [Managed Instance](#).

## NOTE

For more information on migration using the Azure Database Migration Service (DMS), see [Managed Instance migration using DMS](#). For more information on various migration methods, see [SQL Server instance migration to Azure SQL Database Managed Instance](#).

## Prerequisites

This quickstart:

- Uses resources from the [Create a Managed Instance](#) quickstart.
- Requires your computer have the latest [SQL Server Management Studio](#) installed.
- Requires using SSMS to connect to your Managed Instance. See these quickstarts on how to connect:
  - [Enable public endpoint](#) on Managed Instance - this is recommended approach for this tutorial.
  - [Connect to an Azure SQL Database Managed Instance from an Azure VM](#)
  - [Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises](#).

## NOTE

For more information on backing up and restoring a SQL Server database using Azure Blob storage and a [Shared Access Signature \(SAS\) key](#), see [SQL Server Backup to URL](#).

## Restore the database from a backup file

In SSMS, follow these steps to restore the Wide World Importers database to your Managed Instance. The database backup file is stored in a pre-configured Azure Blob storage account.

1. Open SMSS and connect to your Managed Instance.
2. From the left-hand menu, right-click your Managed Instance and select **New Query** to open a new query window.
3. Run the following SQL script, which uses a pre-configured storage account and SAS key to [create a credential](#) in your Managed Instance.

```
CREATE CREDENTIAL [https://mitutorials.blob.core.windows.net/databases]
WITH IDENTITY = 'SHARED ACCESS SIGNATURE'
, SECRET = 'sv=2017-11-09&ss=bfqt&srt=sco&sp=rwdlacup&se=2028-09-06T02:52:55Z&st=2018-09-04T18:52:55Z&spr=https&sig=WOTiM%2FS4GVF%2FEEs9DGQR9Im0W%2BwndxW2CQ7%2B5fHd7Is%3D'
```

The screenshot shows the SSMS interface with a query window titled "SQLQuery2.sql - carl...ster (carlrb (126))". The script creates a credential:

```
CREATE CREDENTIAL [https://mitutorials.blob.core.windows.net/databases]
 WITH IDENTITY = 'SHARED ACCESS SIGNATURE'
 , SECRET = 'sv=2017-11-09&ss=bfqt&srt=sco&sp=rwdlacup&se=2028-09-06T02:52:55Z&st=2018-09-04T18:52:55Z&s'
```

The "Messages" pane shows "Commands completed successfully." and the status bar indicates "Query executed successfully.".

4. To check your credential, run the following script, which uses a [container](#) URL to get a backup file list.

The screenshot shows the SSMS interface with a query window titled "SQLQuery2.sql - carl...ster (carlrb (126))". The command runs:

```
RESTORE FILELISTONLY FROM URL =
'https://mitutorials.blob.core.windows.net/databases/WideWorldImporters-Standard.bak'
```

The "Results" pane displays the backup file list:

LogicalName	PhysicalName	Type	FileGroupName	Size	MaxSize	FileId	CreateLSN	DropLSN	Uniqued
1 WWI_Primary	D:\Data\WideWorldImporters.mdf	D	PRIMARY	1073741824	35184372080640	1	0	0	8D30F4F9-A463-4
2 WWI_UserData	D:\Data\WideWorldImporters_UserData.ndf	D	USERDATA	2147483648	35184372080640	3	37000000095200001	0	28D406E0-78FF-4
3 WWI_Log	E\Log\WideWorldImporters.ldf	L	NULL	104857600	2199023255552	2	0	0	6AC6807E-8774-4

The status bar indicates "Query executed successfully.".

5. Run the following script to restore the Wide World Importers database.

The screenshot shows the SSMS interface with a query window titled "SQLQuery2.sql - carl...ster (carlrb (126))". The command runs:

```
RESTORE DATABASE [Wide World Importers] FROM URL =
'https://mitutorials.blob.core.windows.net/databases/WideWorldImporters-Standard.bak'
```

The "Messages" pane shows "Commands completed successfully." and the status bar indicates "Query executed successfully.".

6. Run the following script to track your restore's status.

The screenshot shows the SSMS interface with a query window titled "SQLQuery2.sql - carl...ster (carlrb (126))". The script selects tracking information:

```
SELECT session_id as SPID, command, a.text AS Query, start_time, percent_complete
 , dateadd(second,estimated_completion_time/1000, getdate()) as estimated_completion_time
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) a
WHERE r.command in ('BACKUP DATABASE','RESTORE DATABASE')
```

7. When the restore completes, view the database in Object Explorer. You can verify that database restore is completed using [sys.dm\\_operation\\_status](#) view.

**NOTE**

Database restore operation is asynchronous and retriable. You might get some error in SQL Server Management Studio if connection breaks or some time-out expires. Azure SQL Database will keep trying to restore database in the background, and you can track the progress of restore using the [sys.dm\\_exec\\_requests](#) and [sys.dm\\_operation\\_status](#) views. In some phases of restore process you will see unique identifier instead of actual database name in the system views. Learn about `RESTORE` statement behavior differences [here](#).

## Next steps

- For troubleshooting a backup to a URL, see [SQL Server Backup to URL Best Practices and Troubleshooting](#).
- For an overview of app connection options, see [Connect your applications to Managed Instance](#).
- To query using your favorite tools or languages, see [Quickstarts: Azure SQL Database Connect and Query](#).

# Tutorial: Migrate SQL Server to an Azure SQL Database managed instance offline using DMS

1/8/2020 • 10 minutes to read • [Edit Online](#)

You can use Azure Database Migration Service to migrate the databases from an on-premises SQL Server instance to an [Azure SQL Database managed instance](#). For additional methods that may require some manual effort, see the article [SQL Server instance migration to Azure SQL Database managed instance](#).

In this tutorial, you migrate the **Adventureworks2012** database from an on-premises instance of SQL Server to a SQL Database managed instance by using Azure Database Migration Service.

In this tutorial, you learn how to:

- Create an instance of Azure Database Migration Service.
- Create a migration project by using Azure Database Migration Service.
- Run the migration.
- Monitor the migration.
- Download a migration report.

## IMPORTANT

For offline migrations from SQL Server to SQL Database managed instance, Azure Database Migration Service can create the backup files for you. Alternately, you can provide the latest full database backup in the SMB network share that the service will use to migrate your databases. Do not append multiple backups into a single backup media; take each backup on a separate backup file. Note that you can use compressed backups as well, to reduce the likelihood of experiencing potential issues with migrating large backups.

## TIP

When you migrate databases to Azure by using Azure Database Migration Service, you can do an *offline* or an *online* migration. With an offline migration, application downtime starts when the migration starts. With an online migration, downtime is limited to the time to cut over at the end of migration. We suggest that you test an offline migration to determine whether the downtime is acceptable; if not, do an online migration.

This article describes an offline migration from SQL Server to a SQL Database managed instance. For an online migration, see [Migrate SQL Server to an Azure SQL Database managed instance online using DMS](#).

## Prerequisites

To complete this tutorial, you need to:

- Create a Microsoft Azure Virtual Network for Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either [ExpressRoute](#) or [VPN](#). [Learn network topologies for Azure SQL Database managed instance migrations using Azure Database Migration Service](#). For more information about creating a virtual network, see the [Virtual Network Documentation](#), and especially the quickstart articles with step-by-step details.

#### **NOTE**

During virtual network setup, if you use ExpressRoute with network peering to Microsoft, add the following service endpoints to the subnet in which the service will be provisioned:

- Target database endpoint (for example, SQL endpoint, Cosmos DB endpoint, and so on)
- Storage endpoint
- Service bus endpoint

This configuration is necessary because Azure Database Migration Service lacks internet connectivity.

- Ensure that your virtual network Network Security Group rules don't block the following inbound communication ports to Azure Database Migration Service: 443, 53, 9354, 445, 12000. For more detail on virtual network NSG traffic filtering, see the article [Filter network traffic with network security groups](#).
- Configure your [Windows Firewall for source database engine access](#).
- Open your Windows Firewall to allow Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433.
- If you're running multiple named SQL Server instances using dynamic ports, you may wish to enable the SQL Browser Service and allow access to UDP port 1434 through your firewalls so that Azure Database Migration Service can connect to a named instance on your source server.
- If you're using a firewall appliance in front of your source databases, you may need to add firewall rules to allow Azure Database Migration Service to access the source database(s) for migration, as well as files via SMB port 445.
- Create an Azure SQL Database managed instance by following the detail in the article [Create an Azure SQL Database managed instance in the Azure portal](#).
- Ensure that the logins used to connect the source SQL Server and target managed instance are members of the sysadmin server role.

#### **NOTE**

By default, Azure Database Migration Service only supports migrating SQL logins. However, you can enable the ability to migrate Windows logins by:

- Ensuring that the target SQL Database managed instance has AAD read access, which can be configured via the Azure portal by a user with the **Company Administrator** or a **Global Administrator** role.
- Configuring your Azure Database Migration Service instance to enable Windows user/group login migrations, which is set up via the Azure portal, on the Configuration page. After enabling this setting, restart the service for the changes to take effect.

After restarting the service, Windows user/group logins appear in the list of logins available for migration. For any Windows user/group logins you migrate, you are prompted to provide the associated domain name. Service user accounts (account with domain name NT AUTHORITY) and virtual user accounts (account name with domain name NT SERVICE) are not supported.

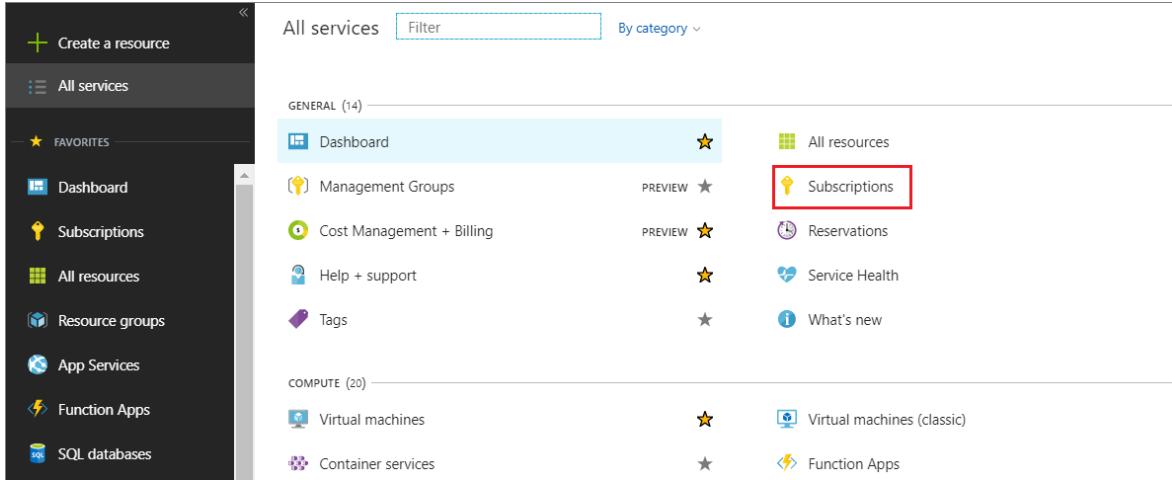
- Create a network share that Azure Database Migration Service can use to back up the source database.
- Ensure that the service account running the source SQL Server instance has write privileges on the network share that you created and that the computer account for the source server has read/write access to the same share.
- Make a note of a Windows user (and password) that has full control privilege on the network share that you previously created. Azure Database Migration Service impersonates the user credential to upload the

backup files to Azure Storage container for restore operation.

- Create a blob container and retrieve its SAS URI by using the steps in the article [Manage Azure Blob Storage resources with Storage Explorer](#), be sure to select all permissions (Read, Write, Delete, List) on the policy window while creating the SAS URI. This detail provides Azure Database Migration Service with access to your storage account container for uploading the backup files used for migrating databases to Azure SQL Database managed instance.

## Register the Microsoft.DataMigration resource provider

1. Sign in to the Azure portal, select **All services**, and then select **Subscriptions**.



2. Select the subscription in which you want to create the instance of Azure Database Migration Service, and then select **Resource providers**.

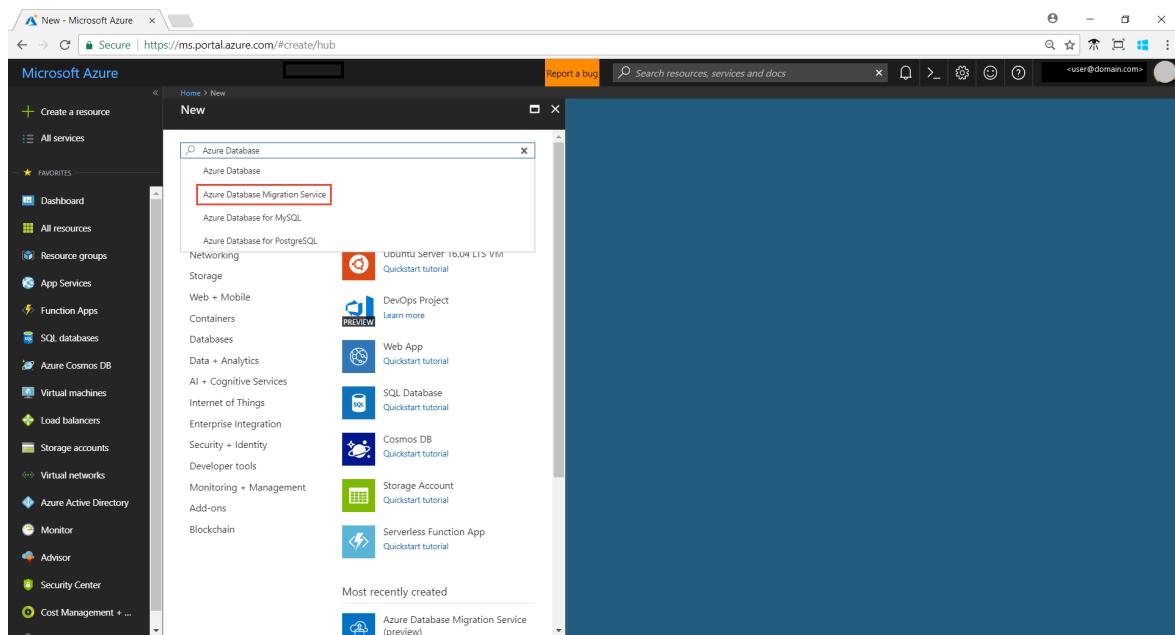
The screenshot shows the Azure portal interface. On the left, there's a navigation sidebar with various service icons like App Services, Function Apps, SQL databases, etc. The main area is titled 'Subscriptions' under 'Microsoft'. It has a search bar and dropdowns for 'My role' (Owner) and 'Status' (3 selected). Below that is a search bar with placeholder 'Search to filter items...'. A table header row shows 'SUBSCRIPTI...' and 'SUBSCRIPTION ID'. The table body contains a single row: '<subscription> <subscription ID>'. To the right of the main content is a sidebar with sections like Overview, Access control (IAM), Diagnose and solve problems, COST MANAGEMENT + BILLING, Partner information, SETTINGS, Programmatic deployment, Resource groups, Resources, Usage + quotas, Policies, Management certificates, My permissions, and Resource providers. The 'Resource providers' section is highlighted with a red box.

3. Search for migration, and then to the right of **Microsoft.DataMigration**, select **Register**.

This screenshot shows the 'Resource providers' blade within the Azure portal. The left side is identical to the previous screenshot, showing the 'Subscriptions' blade. The right side shows a table titled 'Resource providers' with one entry: 'Microsoft.DataMigration' under the 'PROVIDER' column and 'NotRegistered' under the 'STATUS' column. To the right of the status, there is a blue 'Register' button, which is also highlighted with a red box.

## Create an Azure Database Migration Service instance

1. In the Azure portal, select + **Create a resource**, search for **Azure Database Migration Service**, and then select **Azure Database Migration Service** from the drop-down list.



2. On the **Azure Database Migration Service** screen, select **Create**.

The Azure Database Migration Service (DMS) is designed to streamline the process of migrating on-premises databases to Azure. DMS will simplify the migration of existing on-premises SQL Server and Oracle databases to Azure SQL Database, Azure SQL Managed Instance or Microsoft SQL Server in an Azure Virtual Machine. Learn [more](#)

Before using DMS we recommend you complete the following three steps:

1. Open the [Azure Database Migration Guide](#) for step by step guidance through the migration process
2. Assess your SQL Server on-premises database(s) for feature parity and potential compatibility issues by using [Data Migration Assistant \(DMA\)](#). Alternatively, if you are migrating from Oracle, use the [SQL Server Migration Assistant \(SSMA\)](#)
3. Based on your database needs, create a target database using one of the database services: Azure SQL Database, Azure SQL Managed Instance or SQL Server in an Azure Virtual Machine.

Once your assessments are complete, fixes are applied and schema is deployed, proceed with creating a migration service by clicking **Create** below to migrate the data from your source database to the target.

[Save for later](#)

---

PUBLISHER Microsoft

USEFUL LINKS [Documentation](#) [Privacy Statement](#)

**Create**

3. On the **Create Migration Service** screen, specify a name for the service, the subscription, and a new or existing resource group.
4. Select the location in which you want to create the instance of DMS.

5. Select an existing virtual network or create one.

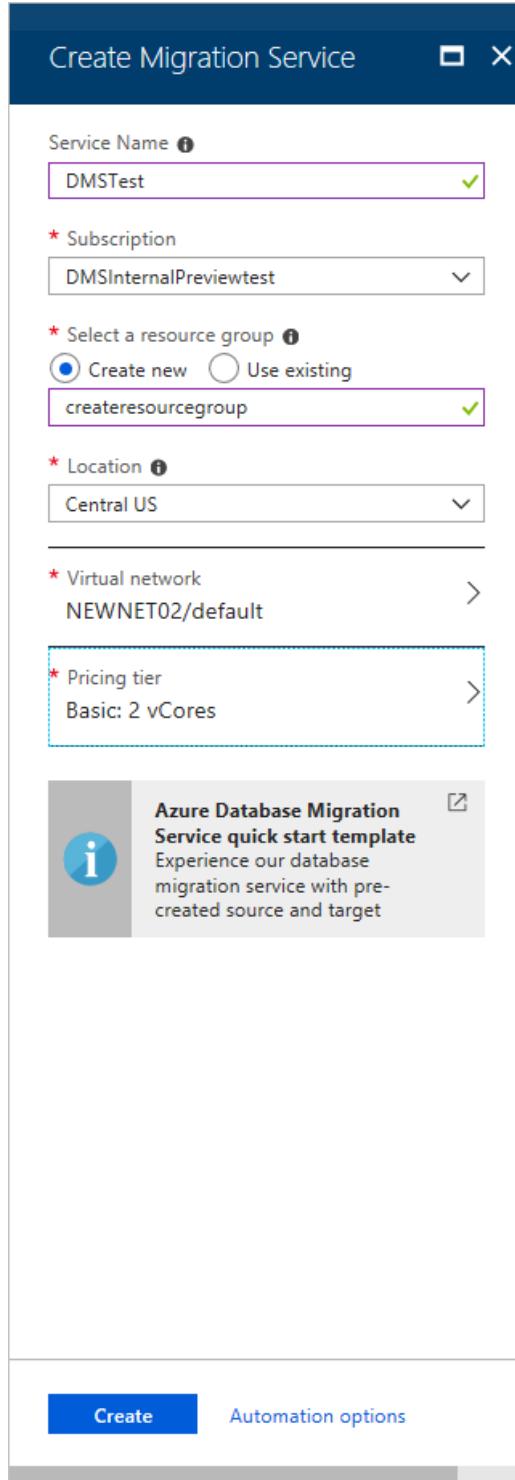
The virtual network provides Azure Database Migration Service with access to the source SQL Server and target Azure SQL Database managed instance.

For more information on how to create a virtual network in Azure portal, see the article [Create a virtual network using the Azure portal](#).

For additional detail, see the article [Network topologies for Azure SQL DB managed instance migrations using Azure Database Migration Service](#).

6. Select a pricing tier.

For more information on costs and pricing tiers, see the [pricing page](#).

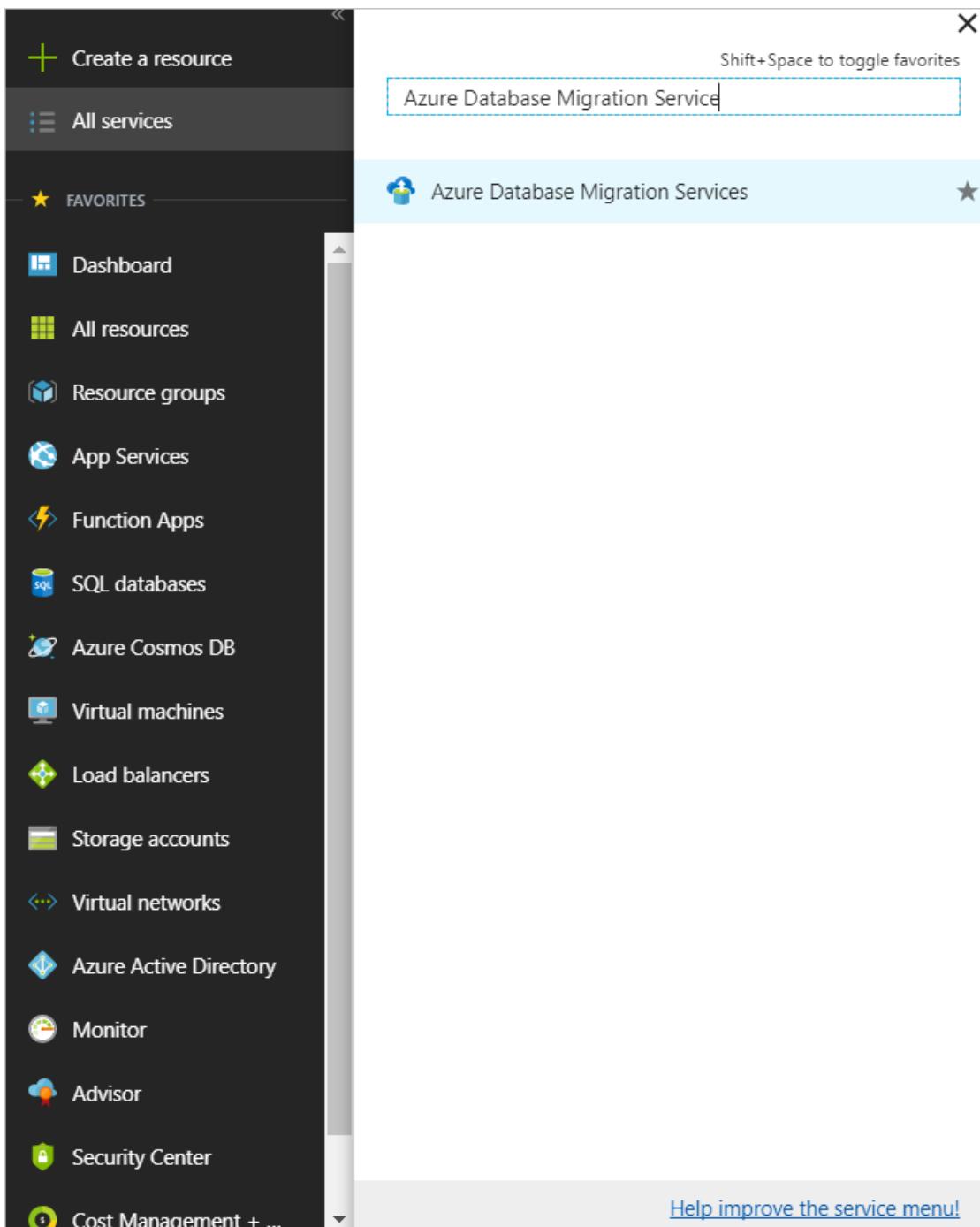


7. Select **Create** to create the service.

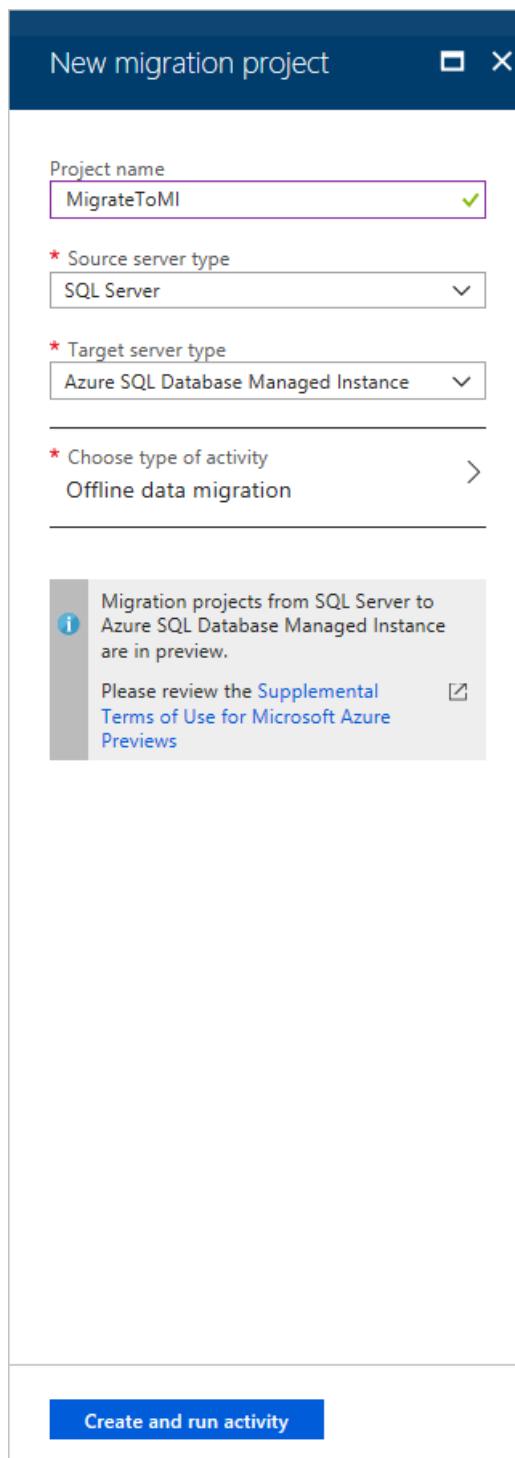
# Create a migration project

After an instance of the service is created, locate it within the Azure portal, open it, and then create a new migration project.

1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.



2. On the **Azure Database Migration Service** screen, search for the name of the instance that you created, and then select the instance.
3. Select + **New Migration Project**.
4. On the **New migration project** screen, specify a name for the project, in the **Source server type** text box, select **SQL Server**, in the **Target server type** text box, select **Azure SQL Database Managed Instance**, and then for **Choose type of activity**, select **Offline data migration**.



5. Select **Create** to create the project.

## Specify source details

1. On the **Migration source detail** screen, specify the connection details for the source SQL Server.
2. If you haven't installed a trusted certificate on your server, select the **Trust server certificate** check box.

When a trusted certificate isn't installed, SQL Server generates a self-signed certificate when the instance is started. This certificate is used to encrypt the credentials for client connections.

**Caution**

SSL connections that are encrypted using a self-signed certificate does not provide strong security. They are susceptible to man-in-the-middle attacks. You should not rely on SSL using self-signed certificates in a production environment or on servers that are connected to the internet.

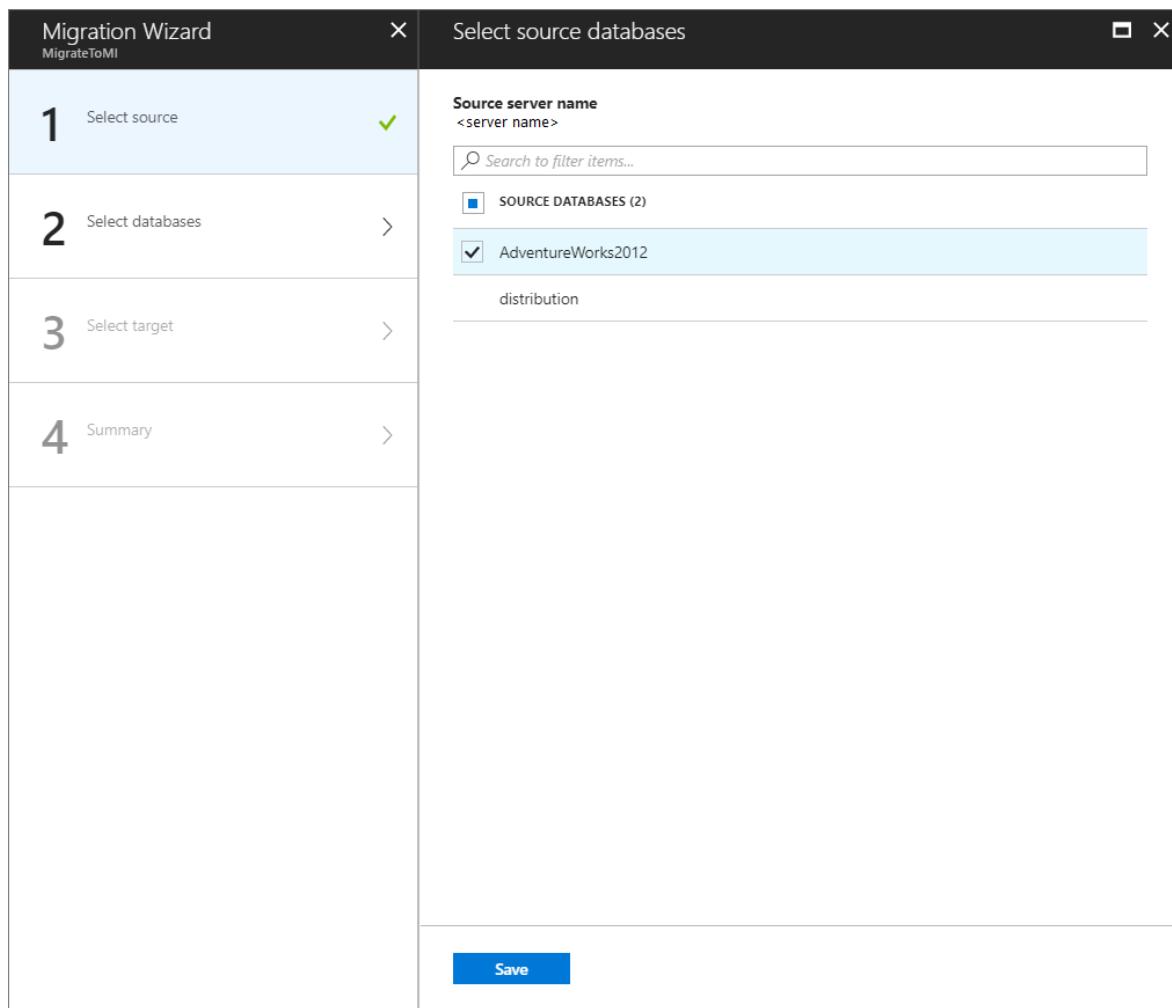
**Migration Wizard**

MigrateToMI

**Source details**

1 Select source	>	* Source SQL Server instance name <small>i</small> Servername.domainname.com
2 Select databases	>	Authentication type SQL Authentication
3 Select target	>	* User Name <small>i</small> Enter user name
4 Summary	>	Password Enter password
Connection properties		
<input checked="" type="checkbox"/> Encrypt connection		
<input type="checkbox"/> Trust server certificate		
<b>Save</b>		

3. Select **Save**.
4. On the **Select source databases** screen, select the **Adventureworks2012** database for migration.



#### IMPORTANT

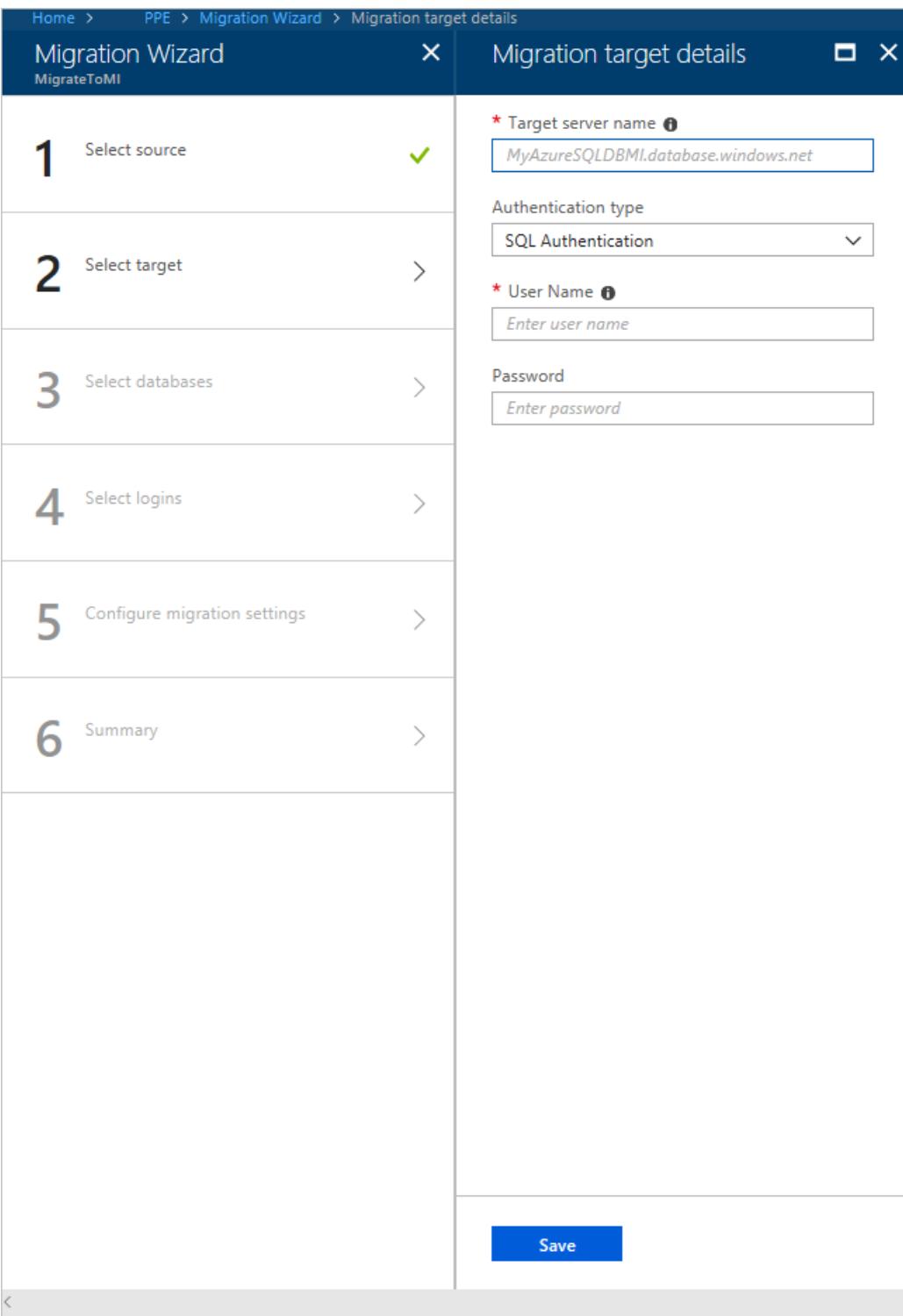
If you use SQL Server Integration Services (SSIS), DMS does not currently support migrating the catalog database for your SSIS projects/packages (SSISDB) from SQL Server to Azure SQL Database managed instance. However, you can provision SSIS in Azure Data Factory (ADF) and redeploy your SSIS projects/packages to the destination SSISDB hosted by Azure SQL Database managed instance. For more information about migrating SSIS packages, see the article [Migrate SQL Server Integration Services packages to Azure](#).

5. Select **Save**.

## Specify target details

1. On the **Migration target details** screen, specify the connection details for the target, which is the pre-provisioned Azure SQL Database managed instance to which you're migrating the **AdventureWorks2012** database.

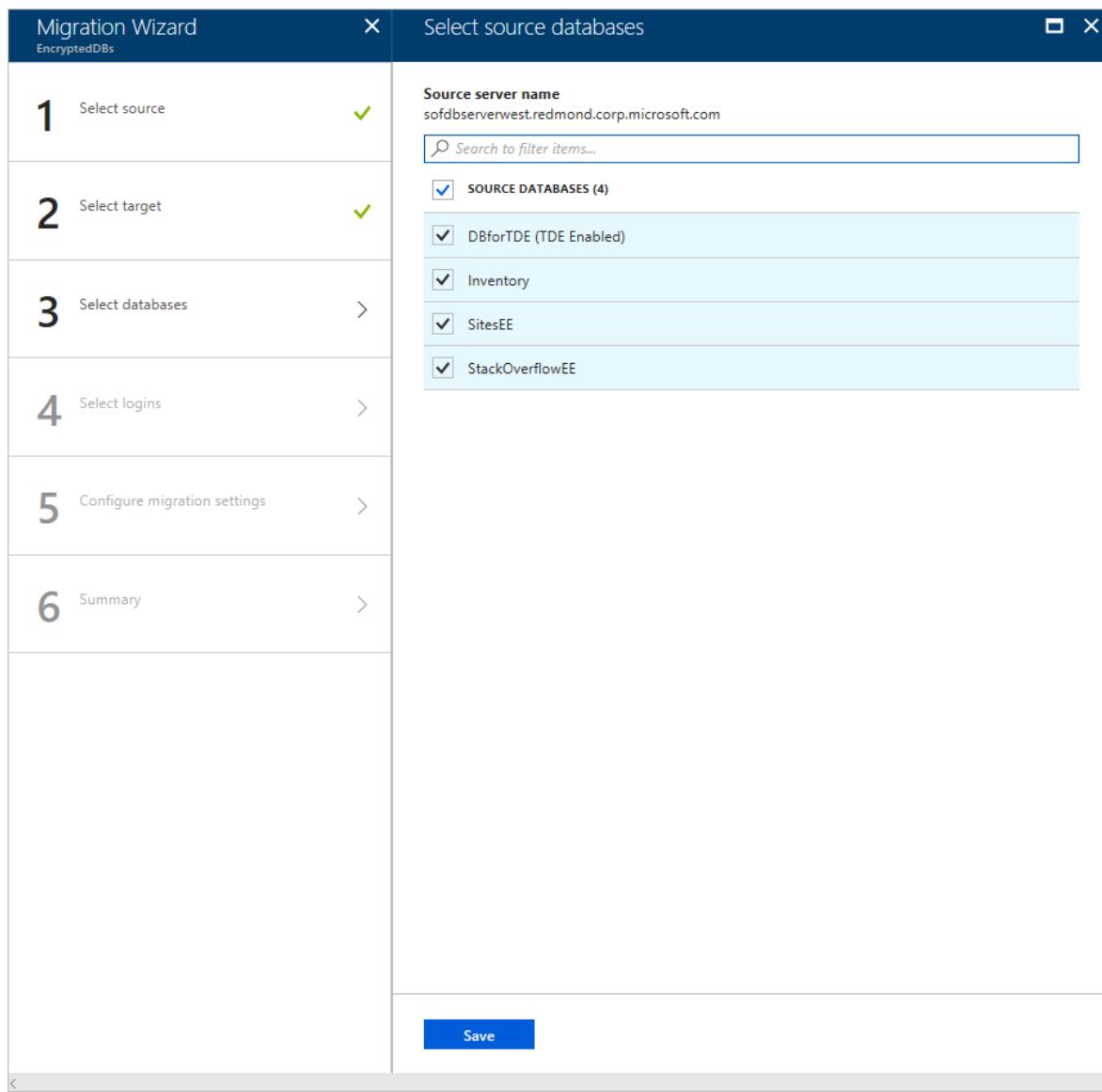
If you haven't already provisioned the SQL Database managed instance, select the [link](#) to help you provision the instance. You can still continue with project creation and then, when the Azure SQL Database managed instance is ready, return to this specific project to execute the migration.



2. Select **Save**.

## Select source databases

1. On the **Select source databases** screen, select the source database that you want to migrate.



2. Select **Save**.

## Select logins

1. On the **Select logins** screen, select the logins that you want to migrate.

### NOTE

By default, Azure Database Migration Service only supports migrating SQL logins. To enable support for migrating Windows logins, see the **Prerequisites** section of this tutorial.

**Select logins**

Source server name  
10.105.129.164

SOURCE LOGINS (28)	LOGIN TYPE	DEFAULT DATABASE	STATUS	READY TO MOVE
NT SERVICE\Winmgmt	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT Service\SQLIaaSExten...	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT SERVICE\SQLSERVER...	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT Service\MSSQLSERVER	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
DemoSQLServer\demouser	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT AUTHORITY\SYSTEM	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
#MS_PolicyTsqlExecutio...	SQL	master	Disabled	Login '#MS_PolicyTsqlExecution.Login##' created by SQL component is not suppor...
sa	SQL	master	Disabled	Login 'sa' is not migrated as it is a system login.
<input checked="" type="checkbox"/> <b>sqluser</b>	SQL	master	Enabled	
<input checked="" type="checkbox"/> <b>distributor_admin</b>	SQL	master	Enabled	
#MS_PolicyEventProces...	SQL	master	Disabled	Login '#MS_PolicyEventProcessingLogin##' created by SQL component is not supp...
NT SERVICE\SQLWriter	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
DEMOQLSERVER\rajpo	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
demouser	SQL	master	Enabled	⚠️ Login already exists, only securables will be migrated and orphan users mapped.
NT SERVICE\Winmgmt	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
DemoSQLServer\demouser	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT SERVICE\SQLSERVER...	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT SERVICESQLWriter	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.
NT_KernelCollisionResolut...	Windows	master	Enabled	LoginType 'WindowsUser' is not supported for migration.

**Save**

2. Select **Save**.

## Configure migration settings

1. On the **Configure migration settings** screen, provide the following detail:

<b>Choose source backup option</b>	Choose the option <b>I will provide latest backup files</b> when you already have full backup files available for DMS to use for database migration. Choose the option <b>I will let Azure Database Migration Service create backup files</b> when you want DMS to take the source database full backup at first and use it for migration.
<b>Network location share</b>	The local SMB network share that Azure Database Migration Service can take the source database backups to. The service account running source SQL Server instance must have write privileges on this network share. Provide an FQDN or IP addresses of the server in the network share, for example, '\\servername.domainname.com\\backupfolder' or '\\IP address\\backupfolder'.
<b>User name</b>	Make sure that the Windows user has full control privilege on the network share that you provided above. Azure Database Migration Service will impersonate the user credential to upload the backup files to Azure Storage container for restore operation. If TDE-enabled databases are selected for migration, the above windows user must be the built-in administrator account and <b>User Account Control</b> must be disabled for Azure Database Migration Service to upload and delete the certificates files.)
<b>Password</b>	Password for the user.

<b>Storage account settings</b>	The SAS URI that provides Azure Database Migration Service with access to your storage account container to which the service uploads the backup files and that is used for migrating databases to Azure SQL Database managed instance. <a href="#">Learn how to get the SAS URI for blob container.</a>
<b>TDE Settings</b>	If you're migrating the source databases with Transparent Data Encryption (TDE) enabled, you need to have write privileges on the target Azure SQL Database managed instance. Select the subscription in which the Azure SQL Database managed instance provisioned from the drop-down menu. Select the target <b>Azure SQL Database Managed Instance</b> in the drop-down menu.

The screenshot shows the Azure Database Migration Wizard interface. On the left, a vertical navigation bar lists steps 1 through 6, with step 5 currently selected and marked with a green checkmark. The main right-hand pane is titled "Configure migration settings" and contains several configuration sections:

- Choose source backup option:** Set to "I will let Azure Database Migration Service create backup files."
- Backup settings:** Includes a note about ensuring the service account has write privileges on the network location share and a field for the "Network share location" (set to "\\\Networkserver.domain.corp.contoso.com").
- Storage account settings:** Includes a note about providing a SAS URI for Azure Storage and a field for the "SAS URI for Azure Storage container" (set to "sig=i7sTHpcH7x%2FqP1J3OHadK5hE%2F%2Fm%2B3%2B%2B%2F%2BpuQYzxls%3D&sr=c").
- TDE Settings:** Includes a note about selecting the target Azure SQL Database Managed Instance and a dropdown for "Select subscription containing the target Azure SQL Database Managed Instance server" (set to "DMSINTERNALDEMO") and "Select target Azure SQL Database Managed Instance" (set to "demomi").

2. Select **Save**.

## Review the migration summary

- On the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity.
- Expand the **Validation option** section to display the **Choose validation option** screen, specify whether to validate the migrated database for query correctness, and then select **Save**.

3. Review and verify the details associated with the migration project.

The screenshot shows two windows side-by-side. On the left is the 'Migration Wizard' window, which lists six steps: 1. Select source, 2. Select target, 3. Select databases, 4. Select logins, 5. Configure migration settings, and 6. Summary. Each step has a green checkmark icon to its right. Step 6 has a right-pointing arrow. On the right is the 'Migration summary' window, which contains the following details:

- Activity name:** demomigration
- Target server name:** demomi.scus1b3fba4c2acae.database.windows.net
- Target server version:** Azure SQL Database Managed Instance 12.0.2000.8
- Source server name:** 10.105.129.164
- Source server version:** SQL Server 2012 11.0.7462.6
- Databases to migrate:** 2 of 5
- Login(s) to migrate:** 6/28
- \* Validation option:** 1/1 options selected

A large blue button at the bottom of the summary window is labeled 'Run migration'.

4. Select **Save**.

## Run the migration

- Select **Run migration**.

The migration activity window appears, and the status of the activity is **Pending**.

## Monitor the migration

1. In the migration activity screen, select **Refresh** to update the display.

Home > DemoDMSService > ReadyMI > MyMigrateToMI

Delete migration Stop migration Refresh Download report

Source server	Target server						
10.105.129.164	demomi.scus1b3fba4c2acae.database.windows.net						
Source version	Target version						
SQL Server 2012	Azure SQL Database Managed Instance						
11.0.7462.6	12.0.2000.8						
Server objects							
3							
<input type="text"/> Search to filter items...							
SERVER OBJ...	NOT START...	IN PROGRESS	COMPLETED	WARNING	FAILED	STOPPED	SKIPPED
Databases	0	0	2	0	0	0	0
Logins	0	0	1	0	0	0	0

You can further expand the databases and logins categories to monitor the migration status of the respective server objects.

MyMigrateToMI

Delete migration Stop migration Refresh Download report

Source server	Target server						
10.105.129.164	demomi.scus1b3fba4c2acae.database.windows.net						
Source version	Target version						
SQL Server 2012	Azure SQL Database Managed Instance						
11.0.7462.6	12.0.2000.8						
Server objects							
3							
<input type="text"/> Search to filter items...							
SERVER OBJ...	NOT START...	IN PROGRESS	COMPLETED	WARNING	FAILED	STOPPED	SKIPPED
Databases	0	0	2	0	0	0	0
Logins	0	0	1	0	0	0	0

MyMigrateToMI

Refresh

Source server	Target server		
10.105.129.164	demomi.scus1b3fba4c2acae.database.windows.net		
Source version	Target version		
SQL Server 2012	Azure SQL Database Managed Instance		
11.0.7462.6	12.0.2000.8		
Logins			
1			
<input type="text"/> Search to filter items...			
NAME	STATUS	MESSAGE	DURATION
demouser	Completed		00:00:01

- After the migration completes, select **Download report** to get a report listing the details associated with the migration process.

3. Verify that the target database on the target Azure SQL Database managed instance environment.

## Next steps

- For a tutorial showing you how to migrate a database to a managed instance using the T-SQL RESTORE command, see [Restore a backup to a managed instance using the restore command](#).
- For information about managed instance, see [What is a managed instance](#).
- For information about connecting apps to a managed instance, see [Connect applications](#).

# Tutorial: Managed instance security in Azure SQL Database using Azure AD server principals (logins)

11/7/2019 • 12 minutes to read • [Edit Online](#)

Managed instance provides nearly all security features that the latest SQL Server on-premises (Enterprise Edition) Database Engine has:

- Limiting access in an isolated environment
- Use authentication mechanisms that require identity (Azure AD, SQL Authentication)
- Use authorization with role-based memberships and permissions
- Enable security features

In this tutorial, you learn how to:

- Create an Azure Active Directory (AD) server principal (login) for a managed instance
- Grant permissions to Azure AD server principals (logins) in a managed instance
- Create Azure AD users from Azure AD server principals (logins)
- Assign permissions to Azure AD users and manage database security
- Use impersonation with Azure AD users
- Use cross-database queries with Azure AD users
- Learn about security features, such as threat protection, auditing, data masking, and encryption

To learn more, see the [Azure SQL Database managed instance overview](#) and [capabilities](#) articles.

## Prerequisites

To complete the tutorial, make sure you have the following prerequisites:

- [SQL Server Management Studio \(SSMS\)](#)
- An Azure SQL Database managed instance
  - Follow this article: [Quickstart: Create an Azure SQL Database managed instance](#)
- Able to access your managed instance and [provisioned an Azure AD administrator for the managed instance](#). To learn more, see:
  - [Connect your application to a managed instance](#)
  - [Managed instance connectivity architecture](#)
  - [Configure and manage Azure Active Directory authentication with SQL](#)

## Limiting access to your managed instance

Managed instances can be accessed through a private IP address. Much like an isolated SQL Server on-premises environment, applications or users need access to the managed instance network (VNet) before a connection can be established. For more information, see the following article, [Connect your application to a managed instance](#).

It is also possible to configure a service endpoint on the managed instance, which allows for public connections, in the same fashion as Azure SQL Database. For more information, see the following article, [Configure public endpoint in Azure SQL Database managed instance](#).

#### NOTE

Even with service endpoints enabled, [SQL Database firewall rules](#) do not apply. Managed instance has its own built-in firewall to manage connectivity.

## Create an Azure AD server principal (login) for a managed instance using SSMS

The first Azure AD server principal (login) can be created by the standard SQL Server account (non-azure AD) that is a `sysadmin`, or the Azure AD admin for the managed instance created during the provisioning process. For more information, see [Provision an Azure Active Directory administrator for your managed instance](#). This functionality has changed since the [GA of Azure AD server principals](#).

See the following articles for examples of connecting to your managed instance:

- [Quickstart: Configure Azure VM to connect to a managed instance](#)
- [Quickstart: Configure a point-to-site connection to a managed instance from on-premises](#)

1. Log into your managed instance using a standard SQL Server account (non-azure AD) that is a `sysadmin` or an Azure AD admin for MI, using [SQL Server Management Studio](#).
2. In **Object Explorer**, right-click the server and choose **New Query**.
3. In the query window, use the following syntax to create a login for a local Azure AD account:

```
USE master
GO
CREATE LOGIN login_name FROM EXTERNAL PROVIDER
GO
```

This example creates a login for the account nativeuser@aadsq1mi.onmicrosoft.com.

```
USE master
GO
CREATE LOGIN [nativeuser@aadsq1mi.onmicrosoft.com] FROM EXTERNAL PROVIDER
GO
```

4. On the toolbar, select **Execute** to create the login.
5. Check the newly added login, by executing the following T-SQL command:

```
SELECT *
FROM sys.server_principals;
GO
```

name	principal_id	sid	type	type_desc
1 nativeuser@aadsq1mi.onmicrosoft.com	327	0x7ABE93DABF8E27479644AB299A5E99FF	E	EXTERNAL_LOGIN

For more information, see [CREATE LOGIN](#).

## Granting permissions to allow the creation of managed instance logins

To create other Azure AD server principals (logins), SQL Server roles or permissions must be granted to the principal (SQL or Azure AD).

## SQL authentication

- If the login is a SQL Principal, only logins that are part of the `sysadmin` role can use the create command to create logins for an Azure AD account.

## Azure AD authentication

- To allow the newly created Azure AD server principal (login) the ability to create other logins for other Azure AD users, groups, or applications, grant the login `sysadmin` or `securityadmin` server role.
- At a minimum, **ALTER ANY LOGIN** permission must be granted to the Azure AD server principal (login) to create other Azure AD server principals (logins).
- By default, the standard permission granted to newly created Azure AD server principals (logins) in master is: **CONNECT SQL** and **VIEW ANY DATABASE**.
- The `sysadmin` server role can be granted to many Azure AD server principals (logins) within a managed instance.

To add the login to the `sysadmin` server role:

1. Log into the managed instance again, or use the existing connection with the Azure AD admin or SQL Principal that is a `sysadmin`.
2. In **Object Explorer**, right-click the server and choose **New Query**.
3. Grant the Azure AD server principal (login) the `sysadmin` server role by using the following T-SQL syntax:

```
ALTER SERVER ROLE sysadmin ADD MEMBER login_name
GO
```

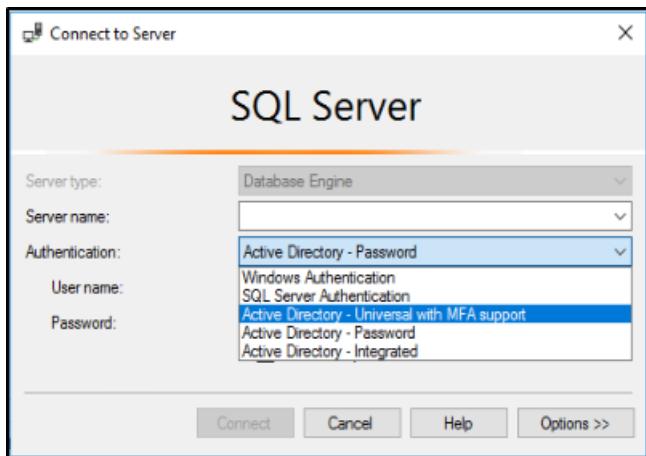
The following example grants the `sysadmin` server role to the login `nativeuser@aadsqmi.onmicrosoft.com`

```
ALTER SERVER ROLE sysadmin ADD MEMBER [nativeuser@aadsqmi.onmicrosoft.com]
GO
```

## Create additional Azure AD server principals (logins) using SSMS

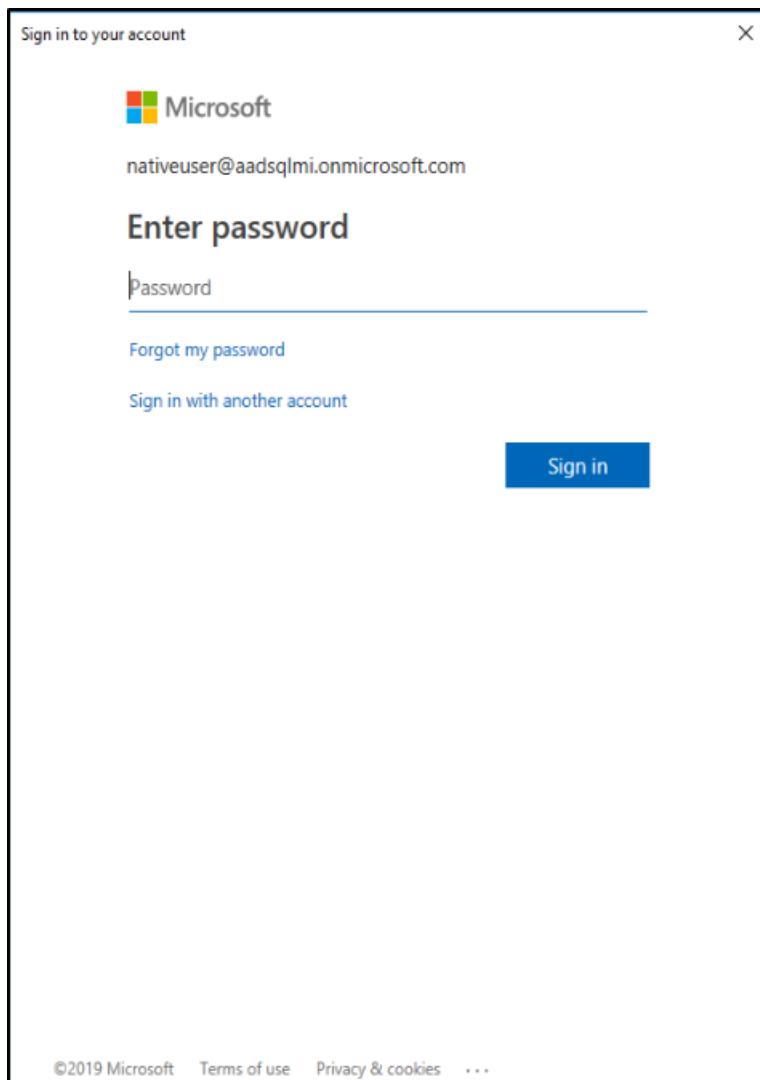
Once the Azure AD server principal (login) has been created, and provided with `sysadmin` privileges, that login can create additional logins using the **FROM EXTERNAL PROVIDER** clause with **CREATE LOGIN**.

1. Connect to the managed instance with the Azure AD server principal (login), using SQL Server Management Studio. Enter your managed instance host name. For Authentication in SSMS, there are three options to choose from when logging in with an Azure AD account:
  - Active Directory - Universal with MFA support
  - Active Directory - Password
  - Active Directory - Integrated



For more information, see the following article: [Universal Authentication with SQL Database and SQL Data Warehouse \(SSMS support for MFA\)](#)

2. Select **Active Directory - Universal with MFA support**. This brings up a Multi-Factor Authentication (MFA) login window. Sign in with your Azure AD password.



3. In SSMS **Object Explorer**, right-click the server and choose **New Query**.
4. In the query window, use the following syntax to create a login for another Azure AD account:

```
USE master
GO
CREATE LOGIN login_name FROM EXTERNAL PROVIDER
GO
```

This example creates a login for the Azure AD user bob@aadsq1mi.net, whose domain aadsq1mi.net is federated with the Azure AD aadsq1mi.onmicrosoft.com.

Execute the following T-SQL command. Federated Azure AD accounts are the managed instance replacements for on-premises Windows logins and users.

```
USE master
GO
CREATE LOGIN [bob@aadsq1mi.net] FROM EXTERNAL PROVIDER
GO
```

5. Create a database in the managed instance using the [CREATE DATABASE](#) syntax. This database will be used to test user logins in the next section.

- a. In **Object Explorer**, right-click the server and choose **New Query**.
  - b. In the query window, use the following syntax to create a database named **MyMITestDB**.

```
CREATE DATABASE MyMITestDB;
GO
```

6. Create a managed instance login for a group in Azure AD. The group will need to exist in Azure AD before you can add the login to managed instance. See [Create a basic group and add members using Azure Active Directory](#). Create a group *mygroup* and add members to this group.
7. Open a new query window in SQL Server Management Studio.

This example assumes there exist a group called *mygroup* in the Azure AD. Execute the following command:

```
USE master
GO
CREATE LOGIN [mygroup] FROM EXTERNAL PROVIDER
GO
```

8. As a test, log into the managed instance with the newly created login or group. Open a new connection to the managed instance, and use the new login when authenticating.
9. In **Object Explorer**, right-click the server and choose **New Query** for the new connection.
10. Check server permissions for the newly created Azure AD server principal (login) by executing the following command:

```
SELECT * FROM sys.fn_my_permissions (NULL, 'DATABASE')
GO
```

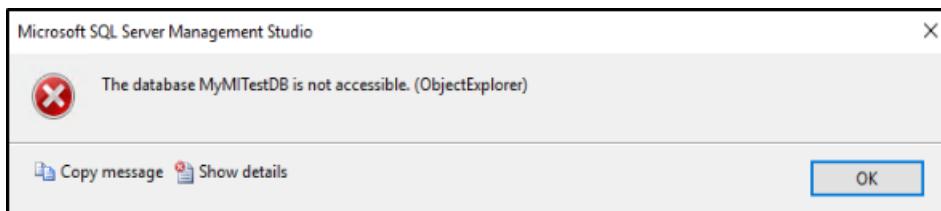
#### NOTE

Azure AD guest users are supported for managed instance logins, only when added as part of an Azure AD Group. An Azure AD guest user is an account that is invited to the Azure AD that the managed instance belongs to, from another Azure AD. For example, joe@contoso.com (Azure AD Account) or steve@outlook.com (MSA Account) can be added to a group in the Azure AD aadsqlmi. Once the users are added to a group, a login can be created in the managed instance **master** database for the group using the **CREATE LOGIN** syntax. Guest users who are members of this group can connect to the managed instance using their current logins (For example, joe@contoso.com or steve@outlook.com).

## Create an Azure AD user from the Azure AD server principal (login) and give permissions

Authorization to individual databases works much in the same way in managed instance as it does with SQL Server on-premises. A user can be created from an existing login in a database, and be provided with permissions on that database, or added to a database role.

Now that we've created a database called **MyMITestDB**, and a login that only has default permissions, the next step is to create a user from that login. At the moment, the login can connect to the managed instance, and see all the databases, but can't interact with the databases. If you sign in with the Azure AD account that has the default permissions, and try to expand the newly created database, you'll see the following error:



For more information on granting database permissions, see [Getting Started with Database Engine Permissions](#).

### Create an Azure AD user and create a sample table

1. Log into your managed instance using a `sysadmin` account using SQL Server Management Studio.
2. In **Object Explorer**, right-click the server and choose **New Query**.
3. In the query window, use the following syntax to create an Azure AD user from an Azure AD server principal (login):

```
USE <Database Name> -- provide your database name
GO
CREATE USER user_name FROM LOGIN login_name
GO
```

The following example creates a user bob@aadsqalmi.net from the login bob@aadsqalmi.net:

```
USE MyMITestDB
GO
CREATE USER [bob@aadsqalmi.net] FROM LOGIN [bob@aadsqalmi.net]
GO
```

4. It's also supported to create an Azure AD user from an Azure AD server principal (login) that is a group.

The following example creates a login for the Azure AD group *mygroup* that exists in your Azure AD.

```
USE MyMITestDB
GO
CREATE USER [mygroup] FROM LOGIN [mygroup]
GO
```

All users that belong to **mygroup** can access the **MyMITestDB** database.

#### IMPORTANT

When creating a **USER** from an Azure AD server principal (login), specify the `user_name` as the same `login_name` from **LOGIN**.

For more information, see [CREATE USER](#).

5. In a new query window, create a test table using the following T-SQL command:

```
USE MyMITestDB
GO
CREATE TABLE TestTable
(
 AccountNum varchar(10),
 City varchar(255),
 Name varchar(255),
 State varchar(2)
);
```

6. Create a connection in SSMS with the user that was created. You'll notice that you cannot see the table **TestTable** that was created by the `sysadmin` earlier. We need to provide the user with permissions to read data from the database.
7. You can check the current permission the user has by executing the following command:

```
SELECT * FROM sys.fn_my_permissions('MyMITestDB', 'DATABASE')
GO
```

#### Add users to database-level roles

For the user to see data in the database, we can provide [database-level roles](#) to the user.

1. Log into your managed instance using a `sysadmin` account using SQL Server Management Studio.
2. In **Object Explorer**, right-click the server and choose **New Query**.
3. Grant the Azure AD user the `db_datareader` database role by using the following T-SQL syntax:

```
Use <Database Name> -- provide your database name
ALTER ROLE db_datareader ADD MEMBER user_name
GO
```

The following example provides the user `bob@aadsqalmi.net` and the group `mygroup` with `db_datareader` permissions on the **MyMITestDB** database:

```
USE MyMITestDB
GO
ALTER ROLE db_datareader ADD MEMBER [bob@aadsq1mi.net]
GO
ALTER ROLE db_datareader ADD MEMBER [mygroup]
GO
```

4. Check the Azure AD user that was created in the database exist by executing the following command:

```
SELECT * FROM sys.database_principals
GO
```

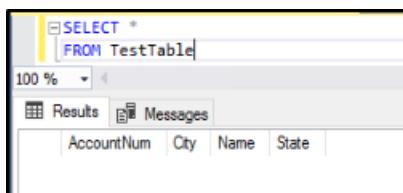
5. Create a new connection to the managed instance with the user that has been added to the `db_datareader` role.
6. Expand the database in **Object Explorer** to see the table.



7. Open a new query window and execute the following SELECT statement:

```
SELECT *
FROM TestTable
```

Are you able to see data from the table? You should see the columns being returned.



## Impersonating Azure AD server-level principals (logins)

Managed instance supports the impersonation of Azure AD server-level principals (logins).

### Test impersonation

1. Log into your managed instance using a `sysadmin` account using SQL Server Management Studio.
2. In **Object Explorer**, right-click the server and choose **New Query**.
3. In the query window, use the following command to create a new stored procedure:

```
USE MyMITestDB
GO
CREATE PROCEDURE dbo.usp_Demo
WITH EXECUTE AS 'bob@aadsq1mi.net'
AS
SELECT user_name();
GO
```

4. Use the following command to see that the user you're impersonating when executing the stored procedure

is **bob@aadsqalmi.net**.

```
Exec dbo.usp_Demo
```

5. Test impersonation by using the EXECUTE AS LOGIN statement:

```
EXECUTE AS LOGIN = 'bob@aadsqalmi.net'
GO
SELECT SUSER_SNAME()
REVERT
GO
```

#### NOTE

Only the SQL server-level principals (logins) that are part of the **sysadmin** role can execute the following operations targeting Azure AD principals:

- EXECUTE AS USER
- EXECUTE AS LOGIN

## Using cross-database queries in managed instances

Cross-database queries are supported for Azure AD accounts with Azure AD server principals (logins). To test a cross-database query with an Azure AD group, we need to create another database and table. You can skip creating another database and table if one already exist.

1. Log into your managed instance using a **sysadmin** account using SQL Server Management Studio.
2. In **Object Explorer**, right-click the server and choose **New Query**.
3. In the query window, use the following command to create a database named **MyMITestDB2** and table named **TestTable2**:

```
CREATE DATABASE MyMITestDB2;
GO
USE MyMITestDB2
GO
CREATE TABLE TestTable2
(
 EmpId varchar(10),
 FirstName varchar(255),
 LastName varchar(255),
 Status varchar(10)
);
```

4. In a new query window, execute the following command to create the user *mygroup* in the new database **MyMITestDB2**, and grant SELECT permissions on that database to *mygroup*:

```
USE MyMITestDB2
GO
CREATE USER [mygroup] FROM LOGIN [mygroup]
GO
GRANT SELECT TO [mygroup]
GO
```

5. Sign into the managed instance using SQL Server Management Studio as a member of the Azure AD group

*mygroup*. Open a new query window and execute the cross-database SELECT statement:

```
USE MyMITestDB
SELECT * FROM MyMITestDB2..TestTable2
GO
```

You should see the table results from **TestTable2**.

## Additional scenarios supported for Azure AD server principals (logins)

- SQL Agent management and job executions are supported for Azure AD server principals (logins).
- Database backup and restore operations can be executed by Azure AD server principals (logins).
- [Auditing](#) of all statements related to Azure AD server principals (logins) and authentication events.
- Dedicated administrator connection for Azure AD server principals (logins) that are members of the `sysadmin` server-role.
- Azure AD server principals (logins) are supported with using the [sqlcmd Utility](#) and [SQL Server Management Studio](#) tool.
- Logon triggers are supported for logon events coming from Azure AD server principals (logins).
- Service Broker and DB mail can be setup using Azure AD server principals (logins).

## Next steps

### Enable security features

See the following [managed instance capabilities security features](#) article for a comprehensive list of ways to secure your database. The following security features are discussed:

- [Managed instance auditing](#)
- [Always encrypted](#)
- [Threat detection](#)
- [Dynamic data masking](#)
- [Row-level security](#)
- [Transparent data encryption \(TDE\)](#)

### Managed instance capabilities

For a complete overview of a managed instance capabilities, see:

### [Managed instance capabilities](#)

# Tutorial: Add a SQL Database managed instance to a failover group

2/18/2020 • 37 minutes to read • [Edit Online](#)

Add a SQL Database managed instance to a failover group. In this article, you will learn how to:

- Create a primary managed instance
- Create a secondary managed instance as part of a [failover group](#).
- Test failover

## NOTE

- When going through this tutorial, ensure you are configuring your resources with the [prerequisites for setting up failover groups for managed instance](#).
- Creating a managed instance can take a significant amount of time. As a result, this tutorial could take several hours to complete. For more information on provisioning times, see [managed instance management operations](#).
- Managed instances participating in a failover group require either [ExpressRoute](#) or two connected VPN gateways. This tutorial provides steps for creating and connecting the VPN gateways. Skip these steps if you already have ExpressRoute configured.

## Prerequisites

- [Portal](#)
- [PowerShell](#)

To complete this tutorial, make sure you have:

- An Azure subscription. [Create a free account](#) if you don't already have one.

## 1 - Create resource group and primary managed instance

In this step, you will create the resource group and the primary managed instance for your failover group using the Azure portal or PowerShell.

- [Portal](#)
- [PowerShell](#)

Create the resource group and your primary managed instance using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the Azure portal. If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select **+ Add** to open the **Select SQL deployment option** page. You can view additional information about the different databases by selecting Show details on the Databases tile.
3. Select **Create** on the **SQL managed instances** tile.

Home > Azure SQL resources > Select SQL deployment option

Select SQL deployment option

How do you plan to use the service?

- SQL databases**  
Best for modern cloud applications. Hyperscale and serverless options are available.  
Resource type: Single database  
[Create](#) [Show details](#)
- SQL managed instances**  
Best for most migrations to the cloud. Lift-and-shift ready.  
Resource type: Single instance  
[Create](#) [Hide details](#)
- SQL virtual machines**  
Best for migrations and applications requiring OS-level access. Lift-and-shift ready.  
Image: Please select an offer  
[Create](#) [Show details](#)

4. On the **Create Azure SQL Database Managed Instance** page, on the **Basics** tab
  - a. Under **Project Details**, select your **Subscription** from the drop-down and then choose to **Create New** resource group. Type in a name for your resource group, such as `myResourceGroup`.
  - b. Under **Managed Instance Details**, provide the name of your managed instance, and the region where you would like to deploy your managed instance. Leave the **Compute + storage** at default values.
  - c. Under **Administrator Account**, provide an admin login, such as `azureuser`, and a complex admin password.

### Create Azure SQL Database Managed Instance

Microsoft

[Basics](#) [Networking](#) [Additional settings](#) [Review + create](#)

SQL Managed Instance is a fully managed PaaS database service with extensive on-premises SQL Server compatibility and native virtual network security. [Learn more](#)

**Project details**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

\* Subscription [?](#)

  \* Resource group [?](#)    
[Create new](#)

**Managed Instance details**

Enter required settings for this instance, including picking a location and configuring the compute and storage resources.

\* Managed Instance name  [?](#) ✓

\* Region  [?](#) ✓

\* Compute + storage [?](#)

**General Purpose**  
Gen5, 8 vCores, 256 GB storage  
[Configure Managed Instance](#)

**Administrator account**

\* Managed Instance admin login  [?](#) ✓

\* Password  [?](#) ✓

\* Confirm password  [?](#) ✓

5. Leave the rest of the settings at default values, and select **Review + create** to review your managed instance settings.

6. Select **Create** to create your primary managed instance.

## 2 - Create secondary virtual network

If you're using the Azure portal to create your managed instance, you will need to create the virtual network separately because there is a requirement that the subnet of the primary and secondary managed instance do not have overlapping ranges. If you're using PowerShell to configure your managed instance, skip ahead to step 3.

- [Portal](#)
- [PowerShell](#)

To verify the subnet range of your primary virtual network, follow these steps:

1. In the [Azure portal](#), navigate to your resource group and select the virtual network for your primary instance.
2. Select **Subnets** under **Settings** and note the **Address range**. The subnet address range of the virtual network for the secondary managed instance cannot overlap this.

NAME	ADDRESS RANGE
ManagedInstance	10.0.0.0/24

To create a virtual network, follow these steps:

1. In the [Azure portal](#), select **Create a resource** and search for *virtual network*.
2. Select the **Virtual Network** option published by Microsoft and then select **Create** on the next page.
3. Fill out the required fields to configure the virtual network for your secondary managed instance, and then select **Create**.

The following table shows the values necessary for the secondary virtual network:

FIELD	VALUE
<b>Name</b>	The name for the virtual network to be used by the secondary managed instance, such as <code>vnet-sql-mi-secondary</code> .

FIELD	VALUE
<b>Address space</b>	The address space for your virtual network, such as 10.128.0.0/16.
<b>Subscription</b>	The subscription where your primary managed instance and resource group reside.
<b>Region</b>	The location where you will deploy your secondary managed instance.
<b>Subnet</b>	The name for your subnet. default is provided for you by default.
<b>Address range</b>	The address range for your subnet. This must be different than the subnet address range used by the virtual network of your primary managed instance, such as 10.128.0.0/24.

Marketplace > Virtual Network > Create virtual network

Create virtual network

\* Name: vnet-sql-mi-secondary ✓

\* Address space ⓘ: 10.128.0.0/16 ✓  
10.128.0.0 - 10.128.255.255 (65536 addresses)

\* Subscription: (dropdown menu)

\* Resource group: myResourceGroup ✓  
Create new

\* Location: (US) East US

Subnet

\* Name: default

\* Address range ⓘ: 10.128.0.0/24 ✓  
10.128.0.0 - 10.128.0.255 (256 addresses)

### 3 - Create a secondary managed instance

In this step, you will create a secondary managed instance in the Azure portal, which will also configure the networking between the two managed instances.

Your second managed instance must:

- Be empty.
- Have a different subnet and IP range than the primary managed instance.

- [Portal](#)
- [PowerShell](#)

Create the secondary managed instance using the Azure portal.

1. Select **Azure SQL** in the left-hand menu of the Azure portal. If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
2. Select **+ Add** to open the **Select SQL deployment option** page. You can view additional information about the different databases by selecting Show details on the Databases tile.
3. Select **Create** on the **SQL managed instances** tile.

4. On the **Basics** tab of the **Create Azure SQL Database Managed Instance** page, fill out the required fields to configure your secondary managed instance.

The following table shows the values necessary for the secondary managed instance:

FIELD	VALUE
<b>Subscription</b>	The subscription where your primary managed instance is.
<b>Resource group</b>	The resource group where your primary managed instance is.
<b>Managed instance name</b>	The name of your new secondary managed instance, such as <code>sql-mi-secondary</code>
<b>Region</b>	The location for your secondary managed instance.
<b>Managed instance admin login</b>	The login you want to use for your new secondary managed instance, such as <code>azureuser</code> .
<b>Password</b>	A complex password that will be used by the admin login for the new secondary managed instance.

5. Under the **Networking** tab, for the **Virtual Network**, select the virtual network you created for the secondary managed instance from the drop-down.

## Create Azure SQL Database Managed Instance

Microsoft

Basics Networking Additional settings Review + create

Configure virtual network and public endpoint connectivity for your Managed Instance. Define level of access and connection type. [Learn more](#)

### Virtual network

Select or create a virtual network to connect to your Managed Instance securely. Allow us to update subnet configuration for you automatically, or follow our guide to set it up yourself. [Learn more](#)

\* Virtual network [?](#) (new) vnet-secondary-sql-mi/ManagedInstance [▼](#)

**i** New virtual network will be created. Network configuration required for Managed Instance will be applied to the subnet automatically.

6. Under the **Additional settings** tab, for **Geo-Replication**, choose to **Yes** to *Use as failover secondary*. Select the primary managed instance from the drop-down.

- a. Be sure that the collation and time zone matches that of the primary managed instance. The primary managed instance created in this tutorial used the default of `SQL_Latin1_General_CI_AS` collation and the `(UTC)` Coordinated Universal Time time zone.

## Create Azure SQL Database Managed Instance

Microsoft

Basics Networking Additional settings Review + create

### Geo-Replication

Use this instance as a Failover Group secondary. [Learn more](#)

\* Use as failover secondary [?](#) [No](#) [Yes](#)

\* Primary Managed Instance [?](#) sql-mi-primary (westus/5f5177a) [▼](#)

**i** Configuration of geo-replication is a multiple step process. This step enables the new instance to join the same DNS zone as the chosen primary instance. Once this instance is created, to complete the configuration create a VPN Gateway between the primary and secondary instance, setup inbound and outbound NSG rules, and create a new auto failover group between the two instances. [Learn more](#)

7. Select **Review + create** to review the settings for your secondary managed instance.

8. Select **Create** to create your secondary managed instance.

## 4 - Create primary gateway

For two managed instances to participate in a failover group, there must be either ExpressRoute or a gateway configured between the virtual networks of the two managed instances to allow network communication. If you choose to configure **ExpressRoute** instead of connecting two VPN gateways, skip ahead to **Step 7**.

This article provides steps to create the two VPN gateways and connect them, but you can skip ahead to creating the failover group if you have configured ExpressRoute instead.

- [Portal](#)
- [PowerShell](#)

Create the gateway for the virtual network of your primary managed instance using the Azure portal.

1. In the [Azure portal](#), go to your resource group and select the **Virtual network** resource for your primary managed instance.

2. Select **Subnets** under **Settings** and then select to add a new **Gateway subnet**. Leave the default values.

The screenshot shows the Azure portal interface for managing subnets. On the left, there's a navigation pane with various options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Address space, Connected devices, and Subnets. The 'Subnets' option is highlighted with a red box. On the right, a modal window titled 'Add subnet' is open. It has fields for 'Name' (set to 'GatewaySubnet') and 'Address range (CIDR block)' (set to '10.0.1.0/24'). Both of these fields are also highlighted with red boxes. Other settings in the modal include Network security group (None), Route table (None), Service endpoints (0 selected), and Subnet delegation (None).

3. Once the subnet gateway is created, select **Create a resource** from the left navigation pane and then type **Virtual network gateway** in the search box. Select the **Virtual network gateway** resource published by **Microsoft**.

The screenshot shows the Azure portal's marketplace search results. The left sidebar has a 'Create a resource' button highlighted with a red box. The main area shows a search bar with 'virtual network gateway' typed into it, also highlighted with a red box. Below the search bar, there's a 'Showing All Results' section. The first item in the list is 'Virtual network gateway Microsoft', which is also highlighted with a red box. To its right is another item, 'Virtual Network Microsoft', with a blue heart icon below it.

4. Fill out the required fields to configure the gateway your primary managed instance.

The following table shows the values necessary for the gateway for the primary managed instance:

FIELD	VALUE
<b>Subscription</b>	The subscription where your primary managed instance is.
<b>Name</b>	The name for your virtual network gateway, such as <code>primary-mi-gateway</code> .
<b>Region</b>	The region where your secondary managed instance is.

FIELD	VALUE
<b>Gateway type</b>	Select <b>VPN</b> .
<b>VPN Type</b>	Select <b>Route-based</b>
<b>SKU</b>	Leave default of <code>VpnGw1</code> .
<b>Location</b>	The location where your primary managed instance and primary virtual network is.
<b>Virtual network</b>	Select the virtual network that was created in section 2, such as <code>vnet-sql-mi-primary</code> .
<b>Public IP address</b>	Select <b>Create new</b> .
<b>Public IP address name</b>	Enter a name for your IP address, such as <code>primary-gateway-IP</code> .

5. Leave the other values as default, and then select **Review + create** to review the settings for your virtual network gateway.

Home > New > Marketplace > Virtual network gateway > Create virtual network gateway

## Create virtual network gateway

\* Subscription

Resource group  myResourceGroup (derived from virtual network's resource group)

**Instance details**

\* Name  primary-mi-gateway ✓

\* Region  (US) West US ✓

\* Gateway type  VPN  ExpressRoute ✓

\* VPN type  Route-based  Policy-based ✓

\* SKU  VpnGw1 ✓

i Only virtual networks in the currently selected subscription and region are listed.

**VIRTUAL NETWORK**

\* Virtual network  vnet-sql-mi-primary ✓  
[Create new](#)

Gateway subnet address range  10.0.1.0/24

**Public IP address**

\* Public IP address  Create new  Use existing ✓

\* Public IP address name  primary-gateway-IP ✓

Public IP address SKU  Basic

\* Assignment  Dynamic  Static ✓

\* Enable active-active mode  Enabled  Disabled ✓

\* Configure BGP ASN  Enabled  Disabled ✓

6. Select **Create** to create your new virtual network gateway.

## 5 - Create secondary gateway

In this step, create the gateway for the virtual network of your secondary managed instance using the Azure portal,

- [Portal](#)
- [PowerShell](#)

Using the Azure portal, repeat the steps in the previous section to create the virtual network subnet and gateway for the secondary managed instance. Fill out the required fields to configure the gateway for your secondary managed instance.

The following table shows the values necessary for the gateway for the secondary managed instance:

FIELD	VALUE
<b>Subscription</b>	The subscription where your secondary managed instance is.
<b>Name</b>	The name for your virtual network gateway, such as <input type="text"/> secondary-mi-gateway.

FIELD	VALUE
<b>Region</b>	The region where your secondary managed instance is.
<b>Gateway type</b>	Select <b>VPN</b> .
<b>VPN Type</b>	Select <b>Route-based</b>
<b>SKU</b>	Leave default of <code>VpnGw1</code> .
<b>Location</b>	The location where your secondary managed instance and secondary virtual network is.
<b>Virtual network</b>	Select the virtual network that was created in section 2, such as <code>vnet-sql-mi-secondary</code> .
<b>Public IP address</b>	Select <b>Create new</b> .
<b>Public IP address name</b>	Enter a name for your IP address, such as <code>secondary-gateway-IP</code> .

## Create virtual network gateway

\* Subscription

Resource group i

myResourceGroup (derived from virtual network's resource group)

### Instance details

\* Name

\* Region

\* Gateway type i
 VPN  ExpressRoute
\* VPN type i
 Route-based  Policy-based
\* SKU i


i Only virtual networks in the currently selected subscription and region are listed.

### VIRTUAL NETWORK

\* Virtual network i

[Create new](#)

Gateway subnet address range

10.128.1.0/24

### Public IP address

\* Public IP address i
 Create new  Use existing

\* Public IP address name

Public IP address SKU

Basic

\* Assignment

 Dynamic  Static
\* Enable active-active mode i
 Enabled  Disabled
\* Configure BGP ASN i
 Enabled  Disabled

## 6 - Connect the gateways

In this step, create a bidirectional connection between the two gateways of the two virtual networks.

- [Portal](#)
- [PowerShell](#)

Connect the two gateways using the Azure portal.

1. Select **Create a resource** from the [Azure portal](#).
2. Type **connection** in the search box and then press enter to search, which takes you to the **Connection** resource, published by Microsoft.
3. Select **Create** to create your connection.
4. On the **Basics** tab, select the following values and then select **OK**.
  - a. Select **vNet-to-VNet** for the **Connection type**.
  - b. Select your subscription from the drop-down.
  - c. Select the resource group for your managed instance in the drop-down.

- d. Select the location of your primary managed instance from the drop-down
5. On the **Settings** tab, select or enter the following values and then select **OK**:
- Choose the primary network gateway for the **First virtual network gateway**, such as `Primary-Gateway`.
  - Choose the secondary network gateway for the **Second virtual network gateway**, such as `Secondary-Gateway`.
  - Select the checkbox next to **Establish bidirectional connectivity**.
  - Either leave the default primary connection name, or rename it to a value of your choice.
  - Provide a **Shared key (PSK)** for the connection, such as `mi1m2psk`.

The screenshot shows the 'Create connection' wizard in the Azure portal. The 'Settings' tab is selected. The process is divided into three steps:

- Step 1: Basics** (Completed): Configure basic settings. Primary-Gateway is selected as the first virtual network gateway.
- Step 2: Settings** (Selected): Configure connection settings. Bidirectional connectivity is established. Connection names are set to Primary-Gateway-to-Secondary-Gateway and Secondary-Gateway-to-Primary-Gateway. A shared key (PSK) mi1m2psk is provided.
- Step 3: Summary** (Not Started): Review and create.

6. On the **Summary** tab, review the settings for your bidirectional connection and then select **OK** to create your connection.

## 7 - Create a failover group

In this step, you will create the failover group and add both managed instances to it.

- [Portal](#)
- [PowerShell](#)

Create the failover group using the Azure portal.

- Select **Azure SQL** in the left-hand menu of the [Azure portal](#). If **Azure SQL** is not in the list, select **All services**, then type Azure SQL in the search box. (Optional) Select the star next to **Azure SQL** to favorite it and add it as an item in the left-hand navigation.
- Select the primary managed instance you created in the first section, such as `sql-mi-primary`.
- Under **Settings**, navigate to **Instance Failover Groups** and then choose to **Add group** to open the **Instance Failover Group** page.

**Microsoft Azure**

Home > Azure SQL resources > sql-mi-primary - Instance Failover Groups (preview)

## sql-mi-primary - Instance Failover Groups (preview)

SQL managed instance

Add group

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Quick start

Connection strings

Active Directory admin

Pricing tier

Instance Failover Groups



The screenshot shows the Microsoft Azure portal interface for managing instance failover groups. On the left, there's a sidebar with various icons and links. A red box highlights the 'Add group' button at the top right of the main content area. Another red box highlights the 'Instance Failover Groups' link in the sidebar.

4. On the **Instance Failover Group** page, type the name of your failover group, such as `failovergrouptutorial` and then choose the secondary managed instance, such as `sql-mi-secondary` from the drop-down. Select **Create** to create your failover group.

**Microsoft Azure**

Home > SQL managed instances > sql-mi-primary

## Instance Failover Group

Create a failover group to manage the automatic failover of managed instances.

Primary managed instance  
sql-mi-primary (westus/General...)

\* Failover group name  
`failovergrouptutorial` ✓

Secondary managed instance ⓘ  
`sql-mi-secondary (eastus/GeneralPurpose)` ▾

Read/Write failover policy  
Automatic

Read/Write grace period (hours)  
1 hours

All databases within your primary managed instance will be replicated to your secondary managed instance.

The screenshot shows the 'Instance Failover Group' configuration page. It includes fields for the failover group name ('failovergrouptutorial') and secondary managed instance ('sql-mi-secondary'). A red box highlights the 'Failover group name' field and the secondary instance dropdown. The 'Primary managed instance' section is also visible.

5. Once failover group deployment is complete, you will be taken back to the **Failover group** page.

## 8 - Test failover

In this step, you will fail your failover group over to the secondary server, and then fail back using the Azure portal.

- [Portal](#)
- [PowerShell](#)

Test failover using the Azure portal.

1. Navigate to your *secondary* managed instance within the [Azure portal](#) and select **Instance Failover Groups** under settings.
2. Review which managed instance is the primary, and which managed instance is the secondary.
3. Select **Failover** and then select **Yes** on the warning about TDS sessions being disconnected.

NAME	PRIMARY MANAGED INSTANCE	SECONDARY MANAGED INSTANCE	READ/WRITE FAILOVER POLICY
failovergrouptutorial	<input checked="" type="checkbox"/> sql-mi-primary (westus)	<input checked="" type="checkbox"/> sql-mi-secondary (eastus)	Automatic

4. Review which managed instance is the primary and which instance is the secondary. If fail over succeeded, the two instances should have switched roles.

NAME	PRIMARY MANAGED INSTANCE	SECONDARY MANAGED INSTANCE	READ/WRITE FAILOVER POLICY
failovergrouptutorial	<input checked="" type="checkbox"/> sql-mi-secondary (eastus)	<input checked="" type="checkbox"/> sql-mi-primary (westus)	Automatic

5. Go to the new *secondary* managed instance and select **Failover** once again to fail the primary instance back to the primary role.

## Clean up resources

Clean up resources by first deleting the managed instance, then the virtual cluster, then any remaining resources, and finally the resource group.

- [Portal](#)
- [PowerShell](#)

1. Navigate to your resource group in the [Azure portal](#).
2. Select the managed instance(s) and then select **Delete**. Type **yes** in the text box to confirm you want to delete the resource and then select **Delete**. This process may take some time to complete in the background, and until it's done, you will not be able to delete the *Virtual cluster* or any other dependent resources. Monitor

the delete in the Activity tab to confirm your managed instance has been deleted.

3. Once the managed instance is deleted, delete the *Virtual cluster* by selecting it in your resource group, and then choosing **Delete**. Type `yes` in the text box to confirm you want to delete the resource and then select **Delete**.
4. Delete any remaining resources. Type `yes` in the text box to confirm you want to delete the resource and then select **Delete**.
5. Delete the resource group by selecting **Delete resource group**, typing in the name of the resource group, `myResourceGroup`, and then selecting **Delete**.

## Full script

- [PowerShell](#)
- [Portal](#)

There are no scripts available for the Azure portal.

## Next steps

In this tutorial, you configured a failover group between two managed instances. You learned how to:

- Create a primary managed instance
- Create a secondary managed instance as part of a [failover group](#).
- Test failover

Advance to the next quickstart on how to connect to your managed instance, and how to restore a database to your managed instance:

[Connect to your managed instance](#) [Restore a database to a managed instance](#)

# Tutorial: Migrate SQL Server on-premises Windows users and groups to Azure SQL Database managed instance using T-SQL DDL syntax

11/20/2019 • 8 minutes to read • [Edit Online](#)

## NOTE

The syntax used to migrate users and groups to managed instance in this article is in **public preview**.

This article takes you through the process of migrating your on-premises Windows users and groups in your SQL Server to an existing Azure SQL Database managed instance using T-SQL syntax.

In this tutorial, you learn how to:

- Create logins for SQL Server
- Create a test database for migration
- Create logins, users, and roles
- Backup and restore your database to managed instance (MI)
- Manually migrate users to MI using ALTER USER syntax
- Testing authentication with the new mapped users

## Prerequisites

To complete this tutorial, the following prerequisites apply:

- The Windows domain is federated with Azure Active Directory (Azure AD).
- Access to Active Directory to create users/groups.
- An existing SQL Server in your on-premises environment.
- An existing managed instance. See [Quickstart: Create an Azure SQL Database managed instance](#).
  - A `sysadmin` in the managed instance must be used to create Azure AD logins.
- [Create an Azure AD admin for managed instance](#).
- You can connect to your managed instance within your network. See the following articles for additional information:
  - [Connect your application to Azure SQL Database managed instance](#)
  - [Quickstart: Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises](#)
  - [Configure public endpoint in Azure SQL Database managed instance](#)

## T-SQL DDL syntax

Below are the T-SQL DDL syntax used to support SQL Server on-premises Windows users and groups migration to managed instance with Azure AD authentication.

```
-- For individual Windows users with logins
ALTER USER [domainName\userName] WITH LOGIN = [loginName@domainName.com];

--For individual groups with logins
ALTER USER [domainName\groupName] WITH LOGIN=[groupName]
```

## Arguments

*domainName*

Specifies the domain name of the user.

*userName*

Specifies the name of the user identified inside the database.

= *loginName@domainName.com*

Remaps a user to the Azure AD login

*groupName*

Specifies the name of the group identified inside the database.

## Part 1: Create logins for SQL Server on-premises users and groups

### IMPORTANT

The following syntax creates a user and a group login in your SQL Server. You'll need to make sure that the user and group exist inside your Active Directory (AD) before executing the below syntax.

Users: testUser1, testGroupUser

Group: migration - testGroupUser needs to belong to the migration group in AD

The example below creates a login in SQL Server for an account named *testUser1* under the domain *aadsqlmi*.

```
-- Sign into SQL Server as a sysadmin or a user that can create logins and databases

use master;
go

-- Create Windows login
create login [aadsqlmi\testUser1] from windows;
go;

/** Create a Windows group login which contains one user [aadsqlmi\testGroupUser].
testGroupUser will need to be added to the migration group in Active Directory
**/
create login [aadsqlmi\migration] from windows;
go;

-- Check logins were created
select * from sys.server_principals;
go;
```

Create a database for this test.

```
-- Create a database called [migration]
create database migration
go
```

## Part 2: Create Windows users and groups, then add roles and permissions

Use the following syntax to create the test user.

```
use migration;
go

-- Create Windows user [aadsqlmi\testUser1] with login
create user [aadsqlmi\testUser1] from login [aadsqlmi\testUser1];
go
```

Check the user permissions:

```
-- Check the user in the Metadata
select * from sys.database_principals;
go

-- Display the permissions - should only have CONNECT permissions
select user_name(grantee_principal_id), * from sys.database_permissions;
go
```

Create a role and assign your test user to this role:

```
-- Create a role with some permissions and assign the user to the role
create role UserMigrationRole;
go

grant CONNECT, SELECT, View DATABASE STATE, VIEW DEFINITION to UserMigrationRole;
go

alter role UserMigrationRole add member [aadsqlmi\testUser1];
go
```

Use the following query to display user names assigned to a specific role:

```
-- Display user name assigned to a specific role
SELECT DP1.name AS DatabaseRoleName,
 ISNULL (DP2.name, 'No members') AS DatabaseUserName
 FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DP1
 ON DRM.role_principal_id = DP1.principal_id
LEFT OUTER JOIN sys.database_principals AS DP2
 ON DRM.member_principal_id = DP2.principal_id
 WHERE DP1.type = 'R'
ORDER BY DP1.name;
```

Use the following syntax to create a group. Then add the group to the role `db_owner`.

```
-- Create Windows group
create user [aadsqlmi\migration] from login [aadsqlmi\migration];
go

-- ADD 'db_owner' role to this group
sp_addrolemember 'db_owner', 'aadsqlmi\migration';
go

--Check the db_owner role for 'aadsqlmi\migration' group
select is_rolemember('db_owner', 'aadsqlmi\migration')
go
-- Output (1 means YES)
```

Create a test table and add some data using the following syntax:

```
-- Create a table and add data
create table test (a int, b int);
go

insert into test values (1,10)
go

-- Check the table values
select * from test;
go
```

## Part 3: Backup and restore the individual user database to managed instance

Create a backup of the migration database using the article [Copy Databases with Backup and Restore](#), or use the following syntax:

```
use master;
go
backup database migration to disk = 'C:\Migration\migration.bak';
go
```

Follow our [Quickstart: Restore a database to a managed instance](#).

## Part 4: Migrate users to managed instance

### NOTE

The Azure AD admin for managed instance functionality after creation has changed. For more information, see [New Azure AD admin functionality for MI](#).

Execute the ALTER USER command to complete the migration process on managed instance.

1. Sign into your managed instance using the Azure AD admin account for managed instance. Then create your Azure AD login in the managed instance using the following syntax. For more information, see [Tutorial: Managed instance security in Azure SQL Database using Azure AD server principals \(logins\)](#).

```

use master
go

-- Create login for AAD user [testUser1@aadsqmi.net]
create login [testUser1@aadsqmi.net] from external provider
go

-- Create login for the Azure AD group [migration]. This group contains one user
[testGroupUser@aadsqmi.net]
create login [migration] from external provider
go

--Check the two new logins
select * from sys.server_principals
go

```

2. Check your migration for the correct database, table, and principals.

```

-- Switch to the database migration that is already restored for MI
use migration;
go

--Check if the restored table test exist and contain a row
select * from test;
go

-- Check that the SQL on-premises Windows user/group exists
select * from sys.database_principals;
go
-- the old user aadsqmi\testUser1 should be there
-- the old group aadsqmi\migration should be there

```

3. Use the ALTER USER syntax to map the on-premises user to the Azure AD login.

```

/** Execute the ALTER USER command to alter the Windows user [aadsqmi\testUser1]
to map to the Azure AD user testUser1@aadsqmi.net
*/
alter user [aadsqmi\testUser1] with login = [testUser1@aadsqmi.net];
go

-- Check the principal
select * from sys.database_principals;
go
-- New user testUser1@aadsqmi.net should be there instead

--Check new user permissions - should only have CONNECT permissions
select user_name(grantee_principal_id), * from sys.database_permissions;
go

-- Check a specific role
-- Display Db user name assigned to a specific role
SELECT DP1.name AS DatabaseRoleName,
isnull (DP2.name, 'No members') AS DatabaseUserName
FROM sys.database_role_members AS DRM
RIGHT OUTER JOIN sys.database_principals AS DP1
ON DRM.role_principal_id = DP1.principal_id
LEFT OUTER JOIN sys.database_principals AS DP2
ON DRM.member_principal_id = DP2.principal_id
WHERE DP1.type = 'R'
ORDER BY DP1.name;

```

4. Use the ALTER USER syntax to map the on-premises group to the Azure AD login.

```

/** Execute ALTER USER command to alter the Windows group [aadsqlmi\migration]
to the Azure AD group login [migration]
*/
alter user [aadsqlmi\migration] with login = [migration];
-- old group migration is changed to Azure AD migration group
go

-- Check the principal
select * from sys.database_principals;
go

--Check the group permission - should only have CONNECT permissions
select user_name(grantee_principal_id), * from sys.database_permissions;
go

--Check the db_owner role for 'aadsqlmi\migration' user
select is_rolemember('db_owner', 'migration')
go
-- Output 1 means 'YES'

```

## Part 5: Testing Azure AD user or group authentication

Test authenticating to managed instance using the user previously mapped to the Azure AD login using the ALTER USER syntax.

1. Log into the federated VM using your MI subscription as `aadsqlmi\testUser1`
2. Using SQL Server Management Studio (SSMS), sign into your managed instance using **Active Directory Integrated** authentication, connecting to the database `migration`.
  - a. You can also sign in using the `testUser1@aadsqlmi.net` credentials with the SSMS option **Active Directory – Universal with MFA support**. However, in this case, you can't use the Single Sign On mechanism and you must type a password. You won't need to use a federated VM to log in to your managed instance.
3. As part of the role member **SELECT**, you can select from the `test` table

```
Select * from test -- and see one row (1,10)
```

Test authenticating to a managed instance using a member of a Windows group `migration`. The user `aadsqlmi\testGroupUser` should have been added to the group `migration` before the migration.

1. Log into the federated VM using your MI subscription as `aadsqlmi\testGroupUser`
2. Using SSMS with **Active Directory Integrated** authentication, connect to the MI server and the database `migration`
  - a. You can also sign in using the `testGroupUser@aadsqlmi.net` credentials with the SSMS option **Active Directory – Universal with MFA support**. However, in this case, you can't use the Single Sign On mechanism and you must type a password. You won't need to use a federated VM to log into your managed instance.
3. As part of the `db_owner` role, you can create a new table.

```
-- Create table named 'new' with a default schema
Create table dbo.new (a int, b int)
```

#### **NOTE**

Due to a known design issue for Azure SQL DB, a create a table statement executed as a member of a group will fail with the following error:

```
Msg 2760, Level 16, State 1, Line 4 The specified schema name "testGroupUser@aadsq1mi.net" either does not exist or you do not have permission to use it.
```

The current workaround is to create a table with an existing schema in the case above <dbo.new>

## Next steps

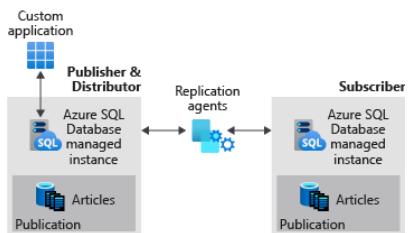
- [Tutorial: Migrate SQL Server to an Azure SQL Database managed instance offline using DMS](#)

# Configure replication in an Azure SQL Database managed instance database

12/23/2019 • 7 minutes to read • [Edit Online](#)

Transactional replication enables you to replicate data into an Azure SQL Database managed instance database from a SQL Server database or another instance database.

This article shows how to configure replication between a managed instance publisher/distributor and a managed instance subscriber.



You can also use transactional replication to push changes made in an instance database in Azure SQL Database managed instance to:

- A SQL Server database.
- A single database in Azure SQL Database.
- A pooled database in an Azure SQL Database elastic pool.

Transactional replication is in public preview on [Azure SQL Database managed instance](#). A managed instance can host publisher, distributor, and subscriber databases. See [transactional replication configurations](#) for available configurations.

## NOTE

- This article is intended to guide a user in configuring replication with an Azure Database managed instance from end to end, starting with creating the resource group. If you already have managed instances deployed, skip ahead to [Step 4](#) to create your publisher database, or [Step 6](#) if you already have a publisher and subscriber database, and are ready to start configuring replication.
- This article configures your publisher and distributor on the same managed instance. To place the distributor on a separate managed instance, see the tutorial [Configure replication between an MI publisher and an MI distributor](#).

## Requirements

Configuring a managed instance to function as a publisher and/or a distributor requires:

- That the managed instance is not currently participating in a geo-replication relationship.
- That the publisher managed instance is on the same virtual network as the distributor and the subscriber, or vNet peering has been established between the virtual networks of all three entities.
- Connectivity uses SQL Authentication between replication participants.
- An Azure Storage Account share for the replication working directory.
- Port 445 (TCP outbound) is open in the security rules of NSG for the managed instances to access the Azure file share. If you encounter the error "failed to connect to azure storage <storage account name> with os error 53", you will need to add an outbound rule to the NSG of the appropriate SQL Managed Instance Subnet.

## NOTE

Single databases and pooled databases in Azure SQL Database can only be subscribers.

## Features

Supports:

- Transactional and snapshot replication mix of SQL Server on-premises and managed instances in Azure SQL Database.
- Subscribers can be in on-premises SQL Server databases, single databases/managed instances in Azure SQL Database, or pooled databases in Azure SQL Database elastic pools.
- One-way or bidirectional replication.

The following features are not supported in a managed instance in Azure SQL Database:

- [Updatable subscriptions](#).
- [Active geo-replication](#) with Transactional replication. Instead of active geo-replication, use [Auto-failover groups](#), but note that the publication has to be [manually deleted](#) from the primary managed instance and recreated on the secondary managed instance after failover.

## 1 - Create a resource group

Use the [Azure portal](#) to create a resource group with the name `SQLMI-Rep1`.

## 2 - Create managed instances

Use the [Azure portal](#) to create two [managed instances](#) on the same virtual network and subnet. For example, name the two managed instances:

- `sql-mi-pub` (along with some characters for randomization)
- `sql-mi-sub` (along with some characters for randomization)

You will also need to [Configure an Azure VM to connect](#) to your Azure SQL Database managed instances.

## 3 - Create Azure Storage Account

[Create an Azure Storage Account](#) for the working directory, and then create a [file share](#) within the storage account.

Copy the file share path in the format of: `\storage-account-name.file.core.windows.net\file-share-name`

Example: `\replstorage.file.core.windows.net\replshare`

Copy the storage access keys in the format of: `DefaultEndpointsProtocol=https;AccountName=<Storage-Account-Name>;AccountKey=****;EndpointSuffix=core.windows.net`

Example:

```
DefaultEndpointsProtocol=https;AccountName=replstorage;AccountKey=dYT5hHZVu9aTgIteGfpYE64cfis0mpKTmmc8+EP53GxuRg6TCwe5eTYWrQM4AmQSG51b3OBskhg==;EndpointSuffix=core.windows.net
```

For more information, see [Manage storage account access keys](#).

## 4 - Create a publisher database

Connect to your `sql-mi-pub` managed instance using SQL Server Management Studio and run the following Transact-SQL (T-SQL) code to create your publisher database:

```
USE [master]
GO

CREATE DATABASE [Rep1Tran_PUB]
GO

USE [Rep1Tran_PUB]
GO
CREATE TABLE Rep1Test (
 ID INT NOT NULL PRIMARY KEY,
 c1 VARCHAR(100) NOT NULL,
 dt1 DATETIME NOT NULL DEFAULT getdate()
)
GO

USE [Rep1Tran_PUB]
GO

INSERT INTO Rep1Test (ID, c1) VALUES (6, 'pub')
INSERT INTO Rep1Test (ID, c1) VALUES (2, 'pub')
INSERT INTO Rep1Test (ID, c1) VALUES (3, 'pub')
INSERT INTO Rep1Test (ID, c1) VALUES (4, 'pub')
INSERT INTO Rep1Test (ID, c1) VALUES (5, 'pub')
GO
SELECT * FROM Rep1Test
GO
```

## 5 - Create a subscriber database

Connect to your `sql-mi-sub` managed instance using SQL Server Management Studio and run the following T-SQL code to create your empty subscriber database:

```
USE [master]
GO

CREATE DATABASE [Rep1Tran_SUB]
GO

USE [Rep1Tran_SUB]
GO
CREATE TABLE Rep1Test (
 ID INT NOT NULL PRIMARY KEY,
 c1 VARCHAR(100) NOT NULL,
 dt1 DATETIME NOT NULL DEFAULT getdate()
)
GO
```

## 6 - Configure distribution

Connect to your `sql-mi-pub` managed instance using SQL Server Management Studio and run the following T-SQL code to configure your distribution database.

```
USE [master]
GO

EXEC sp_adddistributor @distributor = @@ServerName;
EXEC sp_adddistributiondb @database = N'distribution';
GO
```

## 7 - Configure publisher to use distributor

On your publisher managed instance `sql-mi-pub`, change the query execution to [SQLCMD](#) mode and run the following code to register the new distributor with

your publisher.

```
:setvar username loginUsedToAccessSourceManagedInstance
:setvar password passwordUsedToAccessSourceManagedInstance
:setvar file_storage "\\storage-account-name.file.core.windows.net\file-share-name"
-- example: file_storage "\\repilstorage.file.core.windows.net\replshare"
:setvar file_storage_key "DefaultEndpointsProtocol=https;AccountName=<Storage-Account-Name>;AccountKey=****;EndpointSuffix=core.windows.net"
-- example: file_storage_key
"DefaultEndpointsProtocol=https;AccountName=repilstorage;AccountKey=dYT5hHZVu9aTgIteGfpYE64cfis0mpKTmmc8+EP53GxuRg6TCwe5eTYWrQM4AmQSG51b3OBskhg==;EndpointSuffix=core.windows.net"

USE [master]
EXEC sp_adddistpublisher
@publisher = @@ServerName,
@distribution_db = N'distribution',
@security_mode = 0,
@login = N'$username',
@password = N'$password',
@working_directory = N'$(file_storage)',
@storage_connection_string = N'$(file_storage_key)'; -- Remove this parameter for on-premises publishers
```

#### NOTE

Be sure to use only backslashes ( \ ) for the file\_storage parameter. Using a forward slash ( / ) can cause an error when connecting to the file share.

This script configures a local publisher on the managed instance, adds a linked server, and creates a set of jobs for the SQL Server Agent.

## 8 - Create publication and subscriber

Using [SQLCMD](#) mode, run the following T-SQL script to enable replication for your database, and configure replication between your publisher, distributor, and subscriber.

```

-- Set variables
:setvar username sourceLogin
:setvar password sourcePassword
:setvar source_db ReplTran_PUB
:setvar publication_name PublishData
:setvar object ReplTest
:setvar schema dbo
:setvar target_server "sql-mi-sub.wdec33262scj9dr27.database.windows.net"
:setvar target_username targetLogin
:setvar target_password targetPassword
:setvar target_db ReplTran_SUB

-- Enable replication for your source database
USE [$(source_db)]
EXEC sp_replicationdboption
 @dbname = N'$(source_db)',
 @optname = N'publish',
 @value = N'true';

-- Create your publication
EXEC sp_addpublication
 @publication = N'$(publication_name)',
 @status = N'active';

-- Configure your log reader agent
EXEC sp_changelogreader_agent
 @publisher_security_mode = 0,
 @publisher_login = N'$(username)',
 @publisher_password = N'$(password)',
 @job_login = N'$(username)',
 @job_password = N'$(password)';

-- Add the publication snapshot
EXEC sp_addpublication_snapshot
 @publication = N'$(publication_name)',
 @frequency_type = 1,
 @publisher_security_mode = 0,
 @publisher_login = N'$(username)',
 @publisher_password = N'$(password)',
 @job_login = N'$(username)',
 @job_password = N'$(password)';

-- Add the ReplTest table to the publication
EXEC sp_addarticle
 @publication = N'$(publication_name)',
 @type = N'logbased',
 @article = N'$(object)',
 @source_object = N'$(object)',
 @source_owner = N'$(schema)';

-- Add the subscriber
EXEC sp_addsubscription
 @publication = N'$(publication_name)',
 @subscriber = N'$(target_server)',
 @destination_db = N'$(target_db)',
 @subscription_type = N'Push';

-- Create the push subscription agent
EXEC sp_addpushsubscription_agent
 @publication = N'$(publication_name)',
 @subscriber = N'$(target_server)',
 @subscriber_db = N'$(target_db)',
 @subscriber_security_mode = 0,
 @subscriber_login = N'$(target_username)',
 @subscriber_password = N'$(target_password)',
 @job_login = N'$(target_username)',
 @job_password = N'$(target_password)';

-- Initialize the snapshot
EXEC sp_startpublication_snapshot
 @publication = N'$(publication_name)';

```

## 9 - Modify agent parameters

Azure SQL Database managed instance is currently experiencing some backend issues with connectivity with the replication agents. While this issue is being addressed, the workaround to increase the login time out value for the replication agents.

Run the following T-SQL command on the publisher to increase the login timeout:

```
-- Increase login timeout to 150s
update msdb..sysjobsteps set command = command + N' -LoginTimeout 150'
where subsystem in ('Distribution','LogReader','Snapshot') and command not like '%-LoginTimeout %'
```

Run the following T-SQL command again to set the login timeout back to the default value, should you need to do so:

```
-- Increase login timeout to 30
update msdb..sysjobsteps set command = command + N' -LoginTimeout 30'
where subsystem in ('Distribution','LogReader','Snapshot') and command not like '%-LoginTimeout %'
```

Restart all three agents to apply these changes.

## 10 - Test replication

Once replication has been configured, you can test it by inserting new items on the publisher and watching the changes propagate to the subscriber.

Run the following T-SQL snippet to view the rows on the subscriber:

```
select * from dbo.ReplTest
```

Run the following T-SQL snippet to insert additional rows on the publisher, and then check the rows again on the subscriber.

```
INSERT INTO ReplTest (ID, c1) VALUES (15, 'pub')
```

## Clean up resources

To drop the publication, run the following T-SQL command:

```
-- Drops the publication
USE [ReplTran_PUB]
EXEC sp_droppublication @publication = N'PublishData'
GO
```

To remove the replication option from the database, run the following T-SQL command:

```
-- Disables publishing of the database
USE [ReplTran_PUB]
EXEC sp_removedbreplication
GO
```

To disable publishing and distribution, run the following T-SQL command:

```
-- Drops the distributor
USE [master]
EXEC sp_dropdistributor @no_checks = 1
GO
```

You can clean up your Azure resources by [deleting the managed instance resources from the resource group](#) and then deleting the resource group `SQLMI-Rep1`.

## See Also

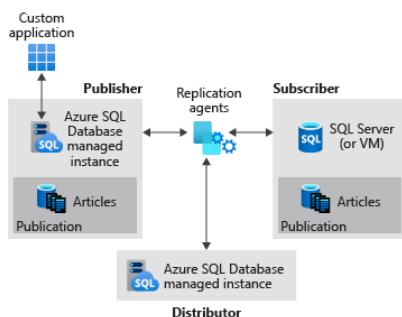
- [Transactional replication](#)
- [Tutorial: Configure transactional replication between an MI publisher and SQL Server subscriber](#)
- [What is a Managed Instance?](#)

# Tutorial: Configure transactional replication between two managed instances and SQL Server

1/23/2020 • 11 minutes to read • [Edit Online](#)

In this tutorial, you learn how to:

- Configure a Managed Instance as a replication Publisher.
- Configure a Managed Instance as a replication Distributor.
- Configure a SQL Server as a subscriber.



This tutorial is intended for an experienced audience and assumes that the user is familiar with deploying and connecting to both managed instances, and SQL Server VMs within Azure. As such, certain steps in this tutorial are glossed over.

To learn more, see the [Azure SQL Database managed instance overview](#), [capabilities](#), and [SQL Transactional Replication](#) articles.

To configure replication between a managed instance publisher and a managed instance subscriber, see [Configure transactional replication between two managed instances](#).

## Prerequisites

To complete the tutorial, make sure you have the following prerequisites:

- An [Azure subscription](#).
- Experience with deploying two managed instances within the same virtual network.
- A SQL Server subscriber, either on-premises or an Azure VM. This tutorial uses an Azure VM.
- [SQL Server Management Studio \(SSMS\) 18.0 or greater](#).
- The latest version of [Azure Powershell](#).
- Port 445, and 1433 allow SQL traffic on both the Azure Firewall and the Windows Firewall.

## 1 - Create the resource group

Use the following PowerShell code snippet to create a new resource group:

```
set variables
$ResourceGroupName = "SQLMI-Repl"
$Location = "East US 2"

Create a new resource group
New-AzResourceGroup -Name $ResourceGroupName -Location $Location
```

## 2 - Create two managed instances

Create two managed instances within this new resource group using the [Azure portal](#).

- The name of the publisher managed instance should be: `sql-mi-publisher` (along with a few characters for randomization) and the name of the virtual network should be `vnet-sql-mi-publisher`.
- The name of the distributor managed instance should be: `sql-mi-distributor` (along with a few characters for randomization) and it should be *in the same virtual network as the publisher managed instance*.

**Create Azure SQL Database Managed Instance**

Microsoft

Basics Networking Additional settings Review + create

Configure virtual network and public endpoint connectivity for your Managed Instance. Define level of access and connection type. [Learn more](#)

**Virtual network**

Select or create a virtual network to connect to your Managed Instance securely. Allow us to update subnet configuration for you automatically, or follow our guide to set it up yourself. [Learn more](#)

Virtual network \* ⓘ (MI) vnet-sql-mi-publisher/ManagedInstance

Selected virtual network was previously configured for Managed Instance. The existing network settings will be retained with no changes.

For more information about creating a managed instance, see [Create a managed instance in the portal](#)

**NOTE**

For the sake of simplicity, and because it is the most common configuration, this tutorial suggests placing the distributor managed instance within the same virtual network as the publisher. However, it's possible to create the distributor in a separate virtual network. To do so, you will need to configure VPN peering between the virtual networks of the publisher and distributor, and then configure VPN peering between the virtual networks of the distributor and subscriber.

### 3 - Create a SQL Server VM

Create a SQL Server virtual machine using the [Azure portal](#). The SQL Server virtual machine should have the following characteristics:

- Name: `sql-vm-sub`
- Image: SQL Server 2016 or greater
- Resource group: the same as the managed instance
- Virtual network: `sql-vm-sub-vnet`

For more information about deploying a SQL Server VM to Azure, see [Quickstart: Create SQL Server VM](#).

### 4 - Configure VPN peering

Configure VPN peering to enable communication between the virtual network of the two managed instances, and the virtual network of SQL Server. To do so, use this PowerShell code snippet:

```
Set variables
$SubscriptionId = '<SubscriptionID>'
$resourceGroup = 'SQLMI-Repl'
$pubvNet = 'sql-mi-publisher-vnet'
$subvNet = 'sql-vm-sub-vnet'
$pubsubName = 'Pub-to-Sub-Peer'
$subpubName = 'Sub-to-Pub-Peer'

$virtualNetwork1 = Get-AzVirtualNetwork `
 -ResourceGroupName $resourceGroup `
 -Name $pubvNet

$virtualNetwork2 = Get-AzVirtualNetwork `
 -ResourceGroupName $resourceGroup `
 -Name $subvNet

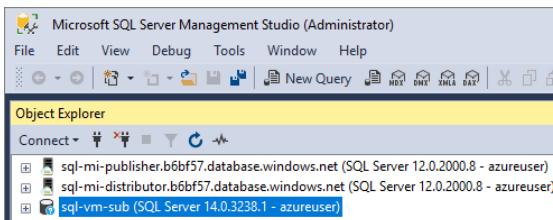
Configure VPN peering from publisher to subscriber
Add-AzVirtualNetworkPeering `
 -Name $pubsubName `
 -VirtualNetwork $virtualNetwork1 `
 -RemoteVirtualNetworkId $virtualNetwork2.Id

Configure VPN peering from subscriber to publisher
Add-AzVirtualNetworkPeering `
 -Name $subpubName `
 -VirtualNetwork $virtualNetwork2 `
 -RemoteVirtualNetworkId $virtualNetwork1.Id

Check status of peering on the publisher vNet; should say connected
Get-AzVirtualNetworkPeering `
 -ResourceGroupName $resourceGroup `
 -VirtualNetworkName $pubvNet `
 | Select PeeringState

Check status of peering on the subscriber vNet; should say connected
Get-AzVirtualNetworkPeering `
 -ResourceGroupName $resourceGroup `
 -VirtualNetworkName $subvNet `
 | Select PeeringState
```

Once VPN peering is established, test connectivity by launching SQL Server Management Studio (SSMS) on your SQL Server and connecting to both managed instances. For more information on connecting to a managed instance using SSMS, see [Use SSMS to connect to the MI](#).



## 5 - Create private DNS zone

A private DNS zone allows DNS routing between the managed instances and the SQL Server.

### Create private DNS Zone

1. Sign into the [Azure portal](#).
2. Select **Create a resource** to create a new Azure resource.
3. Search for `private dns zone` on Azure Marketplace.
4. Choose the **Private DNS zone** resource published by Microsoft and then select **Create** to create the DNS zone.
5. Choose the subscription and resource group from the drop-down.
6. Provide an arbitrary name for your DNS zone such as `repldns.com`.

7. Select **Review + create**. Review the parameters for your private DNS zone and then select **Create** to create your resource.

### Create A record

1. Go to your new **Private DNS zone** and select **Overview**.
2. Select **+ Record set** to create a new A-Record.
3. Provide the name of your SQL Server VM as well as the private internal IP address.

4. Select **OK** to create the A record.

### Link the virtual network

1. Go to your new **Private DNS zone** and select **Virtual network links**.
2. Select **+ Add**.
3. Provide a name for the link, such as `Pub-link`.

4. Select your subscription from the drop-down and then select the virtual network for your publisher managed instance.

5. Check the box next to **Enable auto registration**.

The screenshot shows two side-by-side Azure portal pages. The left page is titled 'repldns.com - Virtual network links' and displays a list of existing links. A red box highlights the '+ Add' button. The right page is titled 'Add virtual network link' and contains fields for 'Link name' (set to 'Pub-link'), 'Virtual network details' (with a note about Resource Manager deployment model), 'Subscription' (selected), 'Virtual network' (selected), and 'Configuration' (with 'Enable auto registration' checked). Both pages have a top navigation bar with 'Home', 'repldns.com - Virtual network links', and 'Add virtual network link'.

6. Select **OK** to link your virtual network.

7. Repeat these steps to add a link for the subscriber virtual network, with a name such as `Sub-link`.

## 6 - Create Azure Storage Account

Create an Azure Storage Account for the working directory, and then create a **file share** within the storage account.

Copy the file share path in the format of: `\storage-account-name.file.core.windows.net\file-share-name`

Example: `\replstorage.file.core.windows.net\replshare`

Copy the storage access key connection string in the format of:

```
DefaultEndpointsProtocol=https;AccountName=<Storage-Account-Name>;AccountKey=****;EndpointSuffix=core.windows.net
```

Example:

```
DefaultEndpointsProtocol=https;AccountName=replstorage;AccountKey=dYT5hHZVu9aTgItGfpYE64cfis0mpKTmmc8+EP53GxuRg6TCwe5eTYWrQM4AmQSG51b3OBskhg==;EndpointSuffix=core.windows.net
```

For more information, see [Manage storage account access keys](#).

## 7 - Create a database

Create a new database on the publisher MI. To do so, follow these steps:

1. Launch SQL Server Management Studio (SSMS) on your SQL Server.
2. Connect to the `sql-mi-publisher` managed instance.
3. Open a **New Query** window and execute the following T-SQL query to create the database:

```
-- Create the databases
USE [master]
GO

-- Drop database if it exists
IF EXISTS (SELECT * FROM sys.sysdatabases WHERE name = 'ReplTutorial')
BEGIN
 DROP DATABASE ReplTutorial
END
GO

-- Create new database
CREATE DATABASE [ReplTutorial]
GO

-- Create table
USE [ReplTutorial]
GO
CREATE TABLE ReplTest (
 ID INT NOT NULL PRIMARY KEY,
 c1 VARCHAR(100) NOT NULL,
 dt1 DATETIME NOT NULL DEFAULT getdate()
)
GO

-- Populate table with data
USE [ReplTutorial]
GO

INSERT INTO ReplTest (ID, c1) VALUES (6, 'pub')
INSERT INTO ReplTest (ID, c1) VALUES (2, 'pub')
INSERT INTO ReplTest (ID, c1) VALUES (3, 'pub')
INSERT INTO ReplTest (ID, c1) VALUES (4, 'pub')
INSERT INTO ReplTest (ID, c1) VALUES (5, 'pub')
GO
SELECT * FROM ReplTest
GO
```

## 8 - Configure distribution

Once connectivity is established and you have a sample database, you can configure distribution on your `sql-mi-distributor` managed instance. To do so, follow these steps:

1. Launch SQL Server Management Studio (SSMS) on your SQL Server.
2. Connect to the `sql-mi-distributor` managed instance.
3. Open a **New Query** window and run the following Transact-SQL code to configure distribution on the distributor managed instance:

```
EXEC sp_adddistpublisher @publisher = 'sql-mi-publisher.b6bf57.database.windows.net', -- primary publisher
 @distribution_db = N'distribution',
 @security_mode = 0,
 @login = N'azureuser',
 @password = N'<publisher_password>',
 @working_directory = N'\\replstorage.file.core.windows.net\replshare',
 @storage_connection_string = N'<storage_connection_string>'
 -- example: @storage_connection_string =
N'DefaultEndpointsProtocol=https;AccountName=replstorage;AccountKey=dYT5hHZVu9aTgIteGfpYE64cfis0mpKTmmc8+EP53GxuRg6TCwe5eTYWrQM4AmQSG5lb3OBskhg==;EndpointSuffix=core.windows.net'
```

### NOTE

Be sure to use only backslashes (`\`) for the `@working_directory` parameter. Using a forward slash (`/`) can cause an error when connecting to the file share.

4. Connect to the `sql-mi-publisher` managed instance.
5. Open a **New Query** window and run the following Transact-SQL code to register the distributor at the publisher:

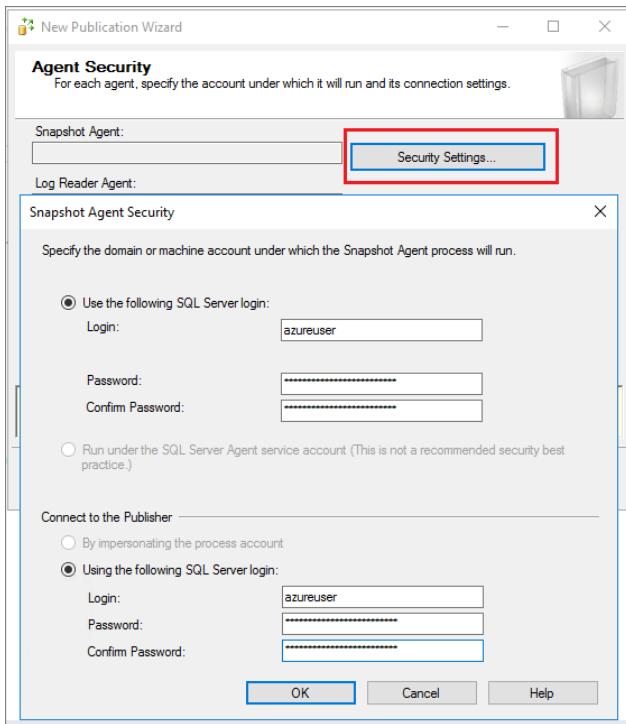
```
Use MASTER
EXEC sys.sp_adddistributor @distributor = 'sql-mi-distributor.b6bf57.database.windows.net', @password = '<distributor_admin_password>'
```

## 9 - Create the publication

Once distribution has been configured, you can now create the publication. To do so, follow these steps:

1. Launch SQL Server Management Studio (SSMS) on your SQL Server.
2. Connect to the `sql-mi-publisher` managed instance.
3. In **Object Explorer**, expand the **Replication** node and right-click the **Local Publication** folder. Select **New Publication...**
4. Select **Next** to move past the welcome page.
5. On the **Publication Database** page, select the `ReplTutorial` database you created previously. Select **Next**.
6. On the **Publication type** page, select **Transactional publication**. Select **Next**.

7. On the **Articles** page, check the box next to **Tables**. Select **Next**.
8. On the **Filter Table Rows** page, select **Next** without adding any filters.
9. On the **Snapshot Agent** page, check the box next to **Create snapshot immediately and keep the snapshot available to initialize subscriptions**. Select **Next**.
10. On the **Agent Security** page, select **Security Settings...**. Provide SQL Server login credentials to use for the Snapshot agent, and to connect to the Publisher. Select **OK** to close the **Snapshot Agent Security** page. Select **Next**.



11. On the **Wizard Actions** page, choose to **Create the publication** and (optionally) choose to **Generate a script file with steps to create the publication** if you want to save this script for later.
12. On the **Complete the Wizard** page, name your publication **ReplTest** and select **Next** to create your publication.
13. Once your publication has been created, refresh the **Replication** node in **Object Explorer** and expand **Local Publications** to see your new publication.

## 10 - Create the subscription

Once the publication has been created, you can create the subscription. To do so, follow these steps:

1. Launch SQL Server Management Studio (SSMS) on your SQL Server.
2. Connect to the **sql-mi-publisher** managed instance.
3. Open a **New Query** window and run the following Transact-SQL code to add the subscription and distribution agent. Use the DNS as part of the subscriber name.

```
use [ReplTutorial]
exec sp_addsubscription
@publication = N'ReplTest',
@subscriber = N'sql-vm-sub.repldns.com', -- include the DNS configured in the private DNS zone
@destination_db = N'ReplSub',
@subscription_type = N'Push',
@sync_type = N'automatic',
@article = N'all',
@update_mode = N'read only',
@subscriber_type = 0

exec sp_addpushsubscription_agent
@publication = N'ReplTest',
@subscriber = N'sql-vm-sub.repldns.com', -- include the DNS configured in the private DNS zone
@subscriber_db = N'ReplSub',
@job_login = N'azureuser',
@job_password = '<Complex Password>',
@subscriber_security_mode = 0,
@subscriber_login = N'azureuser',
@subscriber_password = '<Complex Password>',
@dts_package_location = N'Distributor'
GO
```

## 11 - Test replication

Once replication has been configured, you can test it by inserting new items on the publisher and watching the changes propagate to the subscriber.

Run the following T-SQL snippet to view the rows on the subscriber:

```
Use Rep1Sub
select * from dbo.ReplTest
```

Run the following T-SQL snippet to insert additional rows on the publisher, and then check the rows again on the subscriber.

```
Use ReplTutorial
INSERT INTO ReplTest (ID, c1) VALUES (15, 'pub')
```

## Clean up resources

1. Navigate to your resource group in the [Azure portal](#).
2. Select the managed instance(s) and then select **Delete**. Type `yes` in the text box to confirm you want to delete the resource and then select **Delete**. This process may take some time to complete in the background, and until it's done, you will not be able to delete the *Virtual cluster* or any other dependent resources. Monitor the delete in the Activity tab to confirm your managed instance has been deleted.
3. Once the managed instance is deleted, delete the *Virtual cluster* by selecting it in your resource group, and then choosing **Delete**. Type `yes` in the text box to confirm you want to delete the resource and then select **Delete**.
4. Delete any remaining resources. Type `yes` in the text box to confirm you want to delete the resource and then select **Delete**.
5. Delete the resource group by selecting **Delete resource group**, typing in the name of the resource group, `myResourceGroup`, and then selecting **Delete**.

## Known errors

### Windows logins are not supported

```
Exception Message: Windows logins are not supported in this version of SQL Server.
```

The agent was configured with a Windows login and needs to use a SQL Server login instead. Use the **Agent Security** page of the **Publication properties** to change the login credentials to a SQL Server login.

### Failed to connect to Azure Storage

```
Connecting to Azure Files Storage '\\replstorage.file.core.windows.net\replshare' Failed to connect to Azure Storage '' with OS error: 53.
```

2019-11-19 02:21:05.07 Obtained Azure Storage Connection String for replstorage 2019-11-19 02:21:05.07 Connecting to Azure Files Storage '\\replstorage.file.core.windows.net\replshare' 2019-11-19 02:21:31.21 Failed to connect to Azure Storage '' with OS error: 53.

This is likely because port 445 is closed in either the Azure firewall, the Windows firewall, or both.

```
Connecting to Azure Files Storage '\\replstorage.file.core.windows.net\replshare' Failed to connect to Azure Storage '' with OS error: 55.
```

Using a forward slash instead of backslash in the file path for the file share can cause this error. This is okay: `\\\replstorage.file.core.windows.net\replshare` This can cause an OS 55 error: `'\\replstorage.file.core.windows.net/replshare'`

### Could not connect to Subscriber

```
The process could not connect to Subscriber 'SQL-VM-SUB'. Could not open a connection to SQL Server [53].
```

```
A network-related or instance-specific error has occurred while establishing a connection to SQL Server. Server is not found or not accessible. Check if instance name is correct and if SQL Server is configured to allow remote connections.
```

Possible solutions:

- Ensure port 1433 is open.
- Ensure TCP/IP is enabled on the subscriber.
- Confirm the DNS name was used when creating the subscriber.
- Verify that your virtual networks are correctly linked in the private DNS zone.
- Verify your A-record is configured correctly.
- Verify your VPN peering is configured correctly.

### No publications to which you can subscribe

When adding a new subscription using the **New Subscription** wizard, on the **Publication** page, you may find that there are no databases and publications listed as available options, and you might see the following error message:

```
There are no publications to which you can subscribe, either because this server has no publications or because you do not have sufficient privileges to access the publications.
```

While it's possible that this error message is accurate, and there really aren't publications available on the publisher you connected to, or you're lacking sufficient permissions, this error could also be caused by an older version of SQL Server Management Studio. Try upgrading to SQL Server Management Studio 18.0 or greater to rule this out as a root cause.

## Next steps

### Enable security features

See the following [managed instance capabilities security features](#) article for a comprehensive list of ways to secure your database. The following security features are discussed:

- [Managed instance auditing](#)
- [Always encrypted](#)
- [Threat detection](#)
- [Dynamic data masking](#)
- [Row-level security](#)
- [Transparent data encryption \(TDE\)](#)

## **Managed instance capabilities**

For a complete overview of a managed instance capabilities, see:

[Managed instance capabilities](#)

# Managed API reference for Azure SQL Database Managed Instances

11/7/2019 • 3 minutes to read • [Edit Online](#)

You can create and manage Azure SQL Database Managed Instances using the Azure portal, PowerShell, Azure CLI, REST API, and Transact-SQL. In this article, you can find an overview of functions and API that you can use to create and configure Managed Instance.

## Azure portal: Create a managed instance

For a quickstart showing you how to create an Azure SQL Database Managed Instance, see [Quickstart: Create an Azure SQL Database Managed Instance](#).

## PowerShell: Create and manage managed instances

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

### IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

To create and manage managed instances with Azure PowerShell, use the following PowerShell cmdlets. If you need to install or upgrade PowerShell, see [Install Azure PowerShell module](#).

### TIP

For PowerShell example scripts, see [Quick-start script: Create Azure SQL Managed Instance using PowerShell library](#).

CMDLET	DESCRIPTION
<a href="#">New-AzSqlInstance</a>	Creates an Azure SQL Database Managed Instance
<a href="#">Get-AzSqlInstance</a>	Returns information about Azure SQL Managed Instance
<a href="#">Set-AzSqlInstance</a>	Sets properties for an Azure SQL Database Managed Instance
<a href="#">Remove-AzSqlInstance</a>	Removes an Azure SQL Managed Database Instance
<a href="#">New-AzSqlInstanceDatabase</a>	Creates an Azure SQL Database Managed Instance database

CMDLET	DESCRIPTION
<a href="#">Get-AzSqlInstanceDatabase</a>	Returns information about Azure SQL Managed Instance database
<a href="#">Remove-AzSqlInstanceDatabase</a>	Removes an Azure SQL Managed Database Instance database
<a href="#">Restore-AzSqlInstanceDatabase</a>	Restores an Azure SQL Managed Database Instance database

## Azure CLI: Create and manage managed instances

To create and manage managed instances with [Azure CLI](#), use the following [Azure CLI SQL Managed Instance](#) commands. Use the [Cloud Shell](#) to run the CLI in your browser, or [install](#) it on macOS, Linux, or Windows.

### TIP

For an Azure CLI quickstart, see [Working with SQL Managed Instance using Azure CLI](#).

CMDLET	DESCRIPTION
<code>az sql mi create</code>	Creates a Managed Instance
<code>az sql mi list</code>	Lists available Managed Instances
<code>az sql mi show</code>	Get the details for a Managed Instance
<code>az sql mi update</code>	Updates a Managed Instance
<code>az sql mi delete</code>	Removes a Managed Instance
<code>az sql midb create</code>	Creates a managed database
<code>az sql midb list</code>	Lists available managed databases
<code>az sql midb restore</code>	Restore a managed database
<code>az sql midb delete</code>	Removes a managed database

## Transact-SQL: Create and manage instance databases

To create and manage instance database after the Managed Instance is created, use the following T-SQL commands. You can issue these commands using the Azure portal, [SQL Server Management Studio](#), [Azure Data Studio](#), [Visual Studio Code](#), or any other program that can connect to an Azure SQL Database server and pass Transact-SQL commands.

### TIP

For quickstarts showing you have to configure and connect to a Managed Instance using SQL Server Management Studio on Microsoft Windows, see [Quickstart: Configure Azure VM to connect to an Azure SQL Database Managed Instance](#) and [Quickstart: Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises](#).

**IMPORTANT**

You cannot create or delete a Managed Instance using Transact-SQL.

COMMAND	DESCRIPTION
<a href="#">CREATE DATABASE</a>	Creates a new Managed Instance database. You must be connected to the master database to create a new database.
<a href="#">ALTER DATABASE</a>	Modifies an Azure SQL Managed Instance database.

## REST API: Create and manage managed instances

To create and manage Managed Instances, use these REST API requests.

COMMAND	DESCRIPTION
<a href="#">Managed Instances - Create Or Update</a>	Creates or updates a Managed Instance.
<a href="#">Managed Instances - Delete</a>	Deletes a Managed Instance.
<a href="#">Managed Instances - Get</a>	Gets a Managed Instance.
<a href="#">Managed Instances - List</a>	Returns a list of Managed Instances in a subscription.
<a href="#">Managed Instances - List By Resource Group</a>	Returns a list of Managed Instances in a resource group.
<a href="#">Managed Instances - Update</a>	Updates a Managed Instance.

## Next steps

- To learn about migrating a SQL Server database to Azure, see [Migrate to Azure SQL Database](#).
- For information about supported features, see [Features](#).

# Overview Azure SQL Database managed instance resource limits

2/25/2020 • 9 minutes to read • [Edit Online](#)

This article provides an overview of the technical characteristics and resource limits for Azure SQL Database managed instance, and provides information about how to request an increase to these limits.

## NOTE

For differences in supported features and T-SQL statements see [Feature differences](#) and [T-SQL statement support](#). For general differences between service tiers in single database and managed instance see [Service tier comparison](#).

## Hardware generation characteristics

Managed instance has characteristics and resource limits that depend on the underlying infrastructure and architecture. Azure SQL Database managed instance can be deployed on two hardware generations: Gen4 and Gen5. Hardware generations have different characteristics, as described in the following table:

	GEN4	GEN5
Hardware	Intel E5-2673 v3 (Haswell) 2.4-GHz processors, attached SSD vCore = 1 PP (physical core)	Intel E5-2673 v4 (Broadwell) 2.3-GHz and Intel SP-8160 (Skylake) processors, fast NVMe SSD, vCore=1 LP (hyper-thread)
Number of vCores	8, 16, 24 vCores	4, 8, 16, 24, 32, 40, 64, 80 vCores
Max memory (memory/core ratio)	7 GB per vCore Add more vCores to get more memory.	5.1 GB per vCore Add more vCores to get more memory.
Max In-Memory OLTP memory	Instance limit: 1-1.5 GB per vCore	Instance limit: 0.8 - 1.65 GB per vCore
Max instance reserved storage	General Purpose: 8 TB Business Critical: 1 TB	General Purpose: 8 TB Business Critical 1 TB, 2 TB, or 4 TB depending on the number of cores

## IMPORTANT

- Gen4 hardware is being phased out and is not available anymore for the new deployments. All new managed instances must be deployed on Gen5 hardware.
- Consider [moving your managed instances to Gen 5](#) hardware to experience a wider range of vCore and storage scalability, accelerated networking, best IO performance, and minimal latency.

## In-memory OLTP available space

The amount of In-memory OLTP space in [Business Critical](#) service tier depends on the number of vCores and hardware generation. In the following table are listed limits of memory that can be used for In-memory OLTP objects.

IN-MEMORY OLTP SPACE	GEN5	GEN4
4 vCores	3.14 GB	
8 vCores	6.28 GB	8 GB
16 vCores	15.77 GB	20 GB
24 vCores	25.25 GB	36 GB
32 vCores	37.94 GB	
40 vCores	52.23 GB	
64 vCores	99.9 GB	
80 vCores	131.68 GB	

## Service tier characteristics

Managed instance has two service tiers: [General Purpose](#) and [Business Critical](#). These tiers provide [different capabilities](#), as described in the table below.

### IMPORTANT

Business Critical service-tier provides additional built-in copy of instance (secondary replica) that can be used for read-only workload. If you can separate read-write queries and read-only/analytic/reporting queries, you are getting twice vCores and memory for the same price. Secondary replica might lag few seconds behind the primary instance, so it is designed to offload reporting/analytic workload that don't need exact current state of data. In the table below, **read-only queries** are the queries that are executed on secondary replica.

FEATURE	GENERAL PURPOSE	BUSINESS CRITICAL
Number of vCores*	Gen4: 8, 16, 24 Gen5: 4, 8, 16, 24, 32, 40, 64, 80	Gen4: 8, 16, 24 Gen5: 4, 8, 16, 24, 32, 40, 64, 80 *Same number of vCores is dedicated for read-only queries.
Max memory	Gen4: 56 GB - 168 GB (7GB/vCore) Gen5: 20.4 GB - 408 GB (5.1GB/vCore) Add more vCores to get more memory.	Gen4: 56 GB - 168 GB (7GB/vCore) Gen5: 20.4 GB - 408 GB (5.1GB/vCore) for read-write queries + additional 20.4 GB - 408 GB (5.1GB/vCore) for read-only queries. Add more vCores to get more memory.
Max instance storage size (reserved)	- 2 TB for 4 vCores (Gen5 only) - 8 TB for other sizes	Gen4: 1 TB Gen5: - 1 TB for 4, 8, 16 vCores - 2 TB for 24 vCores - 4 TB for 32, 40, 64, 80 vCores

FEATURE	GENERAL PURPOSE	BUSINESS CRITICAL
Max database size	Up to currently available instance size (max 2 TB - 8 TB depending on the number of vCores).	Up to currently available instance size (max 1 TB - 4 TB depending on the number of vCores).
Max tempDB size	Limited to 24 GB/vCore (96 - 1,920 GB) and currently available instance storage size. Add more vCores to get more TempDB space. Log file size is limited to 120 GB.	Up to currently available instance storage size.
Max number of databases per instance	100, unless the instance storage size limit has been reached.	100, unless the instance storage size limit has been reached.
Max number of database files per instance	Up to 280, unless the instance storage size or <a href="#">Azure Premium Disk storage allocation space</a> limit has been reached.	32,767 files per database, unless the instance storage size limit has been reached.
Max data file size	Limited to currently available instance storage size (max 2 TB - 8 TB) and <a href="#">Azure Premium Disk storage allocation space</a> .	Limited to currently available instance storage size (up to 1 TB - 4 TB).
Max log file size	Limited to 2 TB and currently available instance storage size.	Limited to 2 TB and currently available instance storage size.
Data/Log IOPS (approximate)	Up to 30-40 K IOPS per instance*, 500 - 7500 per file <a href="#">*Increase file size to get more IOPS</a>	10 K - 200 K (2500 IOPS/vCore) Add more vCores to get better IO performance.
Log write throughput limit (per instance)	3 MB/s per vCore Max 22 MB/s	4 MB/s per vCore Max 48 MB/s
Data throughput (approximate)	100 - 250 MB/s per file <a href="#">*Increase the file size to get better IO performance</a>	Not limited.
Storage IO latency (approximate)	5-10 ms	1-2 ms
In-memory OLTP	Not supported	Available, <a href="#">size depends on number of vCore</a>
Max sessions	30000	30000
<a href="#">Read-only replicas</a>	0	1 (included in price)

#### NOTE

- **Currently available instance storage size** is the difference between reserved instance size and the used storage space.
- Both data and log file size in the user and system databases are included in the instance storage size that is compared with the Max storage size limit. Use [sys.master\\_files](#) system view to determine the total used space by databases. Error logs are not persisted and not included in the size. Backups are not included in storage size.
- Throughput and IOPS on General Purpose tier also depend on the [file size](#) that is not explicitly limited by managed instance.
- You can create another readable replica in different Azure region using Auto-failover groups.
- Max instance IOPS depend on the file layout and distribution of workload. As an example, if you create 7 x 1TB files with max 5K IOPS each and 7 small files (smaller than 128 GB) with 500 IOPS each, you can get 38500 IOPS per instance ( $7 \times 5000 + 7 \times 500$ ) if your workload can use all files. Note that some amount of IOPS is also used for auto-backups.

#### NOTE

Find more information about the [resource limits in managed instance pools](#) in this article.

### File IO characteristics in General Purpose tier

In General Purpose service tier every database file is getting dedicated IOPS and throughput that depends on the file size. Bigger files are getting more IOPS and throughput. IO characteristics of the database files are shown in the following table:

FILE SIZE	>=0 AND <=128 GiB	>128 AND <=256 GiB	>256 AND <= 512 GiB	>0.5 AND <=1 TiB	>1 AND <=2 TiB	>2 AND <=4 TiB	>4 AND <=8 TiB
IOPS per file	500	1100	2300	5000	7500	7500	12,500
Throughput per file	100 MiB/s	125 MiB/s	150 MiB/s	200 MiB/s	250 MiB/s	250 MiB/s	480 MiB/s

If you notice high IO latency on some database file or you see that IOPS/throughput is reaching the limit, you might improve performance by [increasing the file size](#).

There are also instance-level limits like max log write throughput 22 MB/s, so you might not be able to reach file throughout on log file because you are reaching instance throughput limit.

## Supported regions

Managed instances can be created only in [supported regions](#). To create a managed instance in a region that is currently not supported, you can [send a support request via the Azure portal](#).

## Supported subscription types

Managed instance currently supports deployment only on the following types of subscriptions:

- [Enterprise Agreement \(EA\)](#)
- [Pay-as-you-go](#)
- [Cloud Service Provider \(CSP\)](#)
- [Enterprise Dev/Test](#)
- [Pay-As-You-Go Dev/Test](#)

- Subscriptions with monthly Azure credit for Visual Studio subscribers

## Regional resource limitations

Supported subscription types can contain a limited number of resources per region. Managed instance has two default limits per Azure region (that can be increased on-demand by creating a special [support request in the Azure portal](#) depending on a type of subscription type:

- **Subnet limit:** The maximum number of subnets where managed instances are deployed in a single region.
- **vCore unit limit:** The maximum number of vCore units that can be deployed across all instances in a single region. One GP vCore uses one vCore unit and one BC vCore takes 4 vCore units. The total number of instances is not limited as long as it is within the vCore unit limit.

### NOTE

These limits are default settings and not technical limitations. The limits can be increased on-demand by creating a special [support request in the Azure portal](#) if you need more managed instances in the current region. As an alternative, you can create new managed instances in another Azure region without sending support requests.

The following table shows the **default regional limits** for supported subscription types (default limits can be extended using support request described below):

SUBSCRIPTION TYPE	MAX NUMBER OF MANAGED INSTANCE SUBNETS	MAX NUMBER OF VCORE UNITS*
Pay-as-you-go	3	320
CSP	8 (15 in some regions**)	960 (1440 in some regions**)
Pay-as-you-go Dev/Test	3	320
Enterprise Dev/Test	3	320
EA	8 (15 in some regions**)	960 (1440 in some regions**)
Visual Studio Enterprise	2	64
Visual Studio Professional and MSDN Platforms	2	32

\* In planning deployments, please take into consideration that Business Critical (BC) service tier requires four (4) times more vCore capacity than General Purpose (GP) service tier. For example: 1 GP vCore = 1 vCore unit and 1 BC vCore = 4 vCore units. To simplify your consumption analysis against the default limits, summarize the vCore units across all subnets in the region where managed instances are deployed and compare the results with the instance unit limits for your subscription type. **Max number of vCore units** limit applies to each subscription in a region. There is no limit per individual subnets except that the sum of all vCores deployed across multiple subnets must be lower or equal to **max number of vCore units**.

\*\* Larger subnet and vCore limits are available in the following regions: Australia East, East US, East US 2, North Europe, South Central US, Southeast Asia, UK South, West Europe, West US 2.

## Request a quota increase for SQL managed instance

If you need more managed instances in your current regions, send a support request to extend the quota using

the Azure portal. For more information, see [Request quota increases for Azure SQL Database](#).

## Next steps

- For more information about managed instance, see [What is a managed instance?](#).
- For pricing information, see [SQL Database managed instance pricing](#).
- To learn how to create your first managed instance, see [the quickstart guide](#).

# Connectivity architecture for a managed instance in Azure SQL Database

1/10/2020 • 14 minutes to read • [Edit Online](#)

This article explains communication in an Azure SQL Database managed instance. It also describes connectivity architecture and how the components direct traffic to the managed instance.

The SQL Database managed instance is placed inside the Azure virtual network and the subnet that's dedicated to managed instances. This deployment provides:

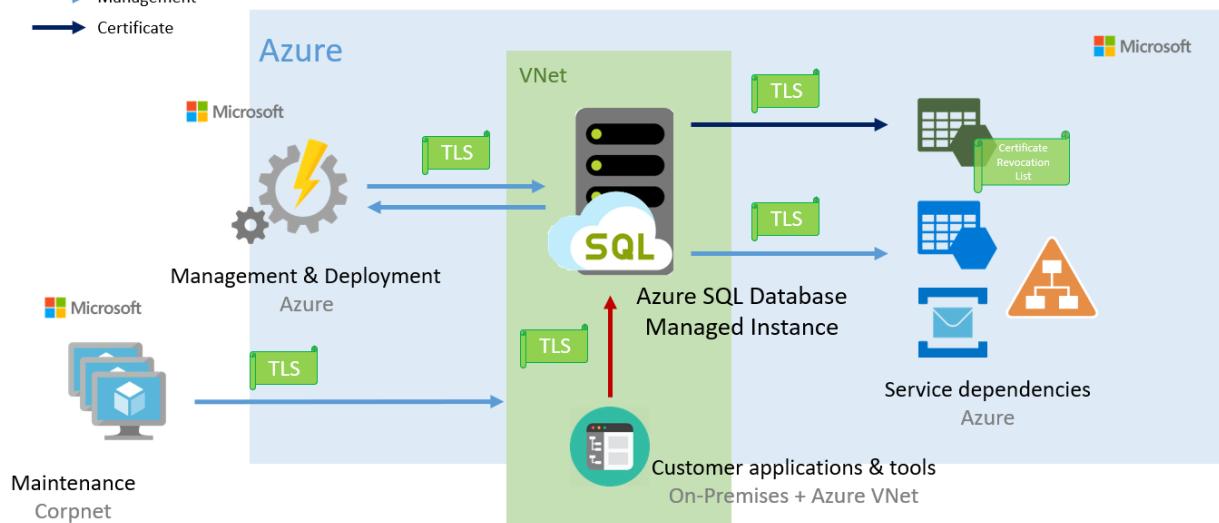
- A secure private IP address.
- The ability to connect an on-premises network to a managed instance.
- The ability to connect a managed instance to a linked server or another on-premises data store.
- The ability to connect a managed instance to Azure resources.

## Communication overview

The following diagram shows entities that connect to a managed instance. It also shows the resources that need to communicate with the managed instance. The communication process at the bottom of the diagram represents customer applications and tools that connect to the managed instance as data sources.

### Legend:

- Data
- Management
- Certificate



A managed instance is a platform as a service (PaaS) offering. Microsoft uses automated agents (management, deployment, and maintenance) to manage this service based on telemetry data streams. Because Microsoft is responsible for management, customers can't access the managed instance virtual cluster machines through Remote Desktop Protocol (RDP).

Some SQL Server operations started by end users or applications might require managed instances to interact with the platform. One case is the creation of a managed instance database. This resource is exposed through the Azure portal, PowerShell, Azure CLI, and the REST API.

Managed instances depend on Azure services such as Azure Storage for backups, Azure Event Hubs for telemetry, Azure Active Directory for authentication, Azure Key Vault for Transparent Data Encryption (TDE) and a couple of Azure platform services that provide security and supportability features. The managed

instances makes connections to these services.

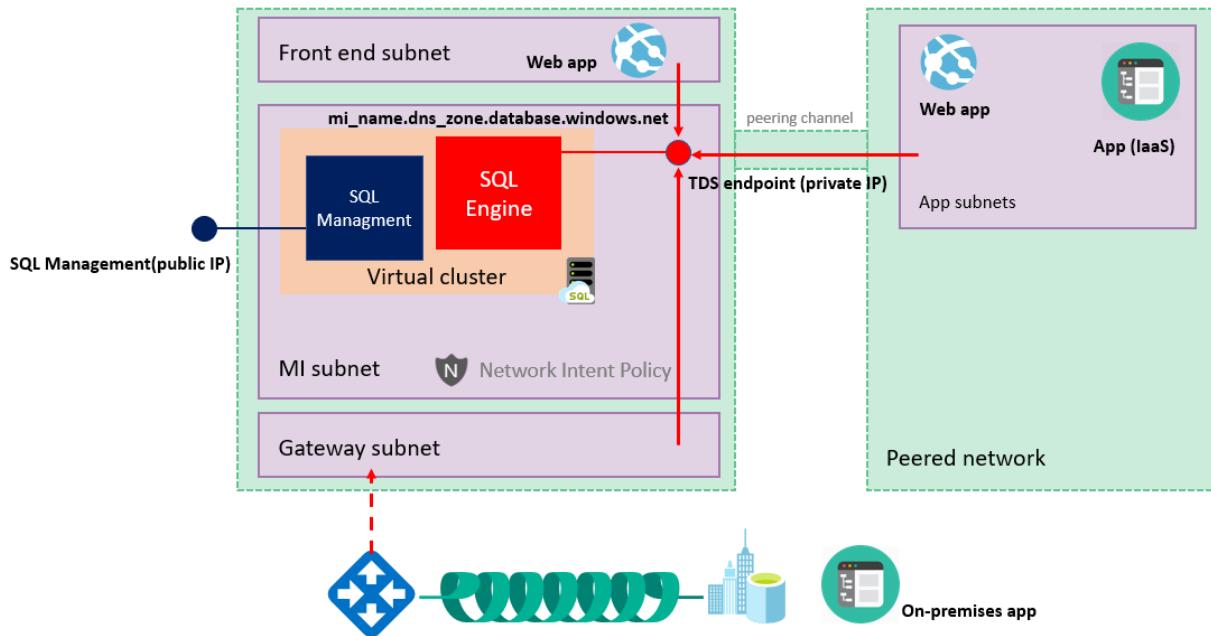
All communications are encrypted and signed using certificates. To check the trustworthiness of communicating parties, managed instances constantly verify these certificates through certificate revocation lists. If the certificates are revoked, the managed instance closes the connections to protect the data.

## High-level connectivity architecture

At a high level, a managed instance is a set of service components. These components are hosted on a dedicated set of isolated virtual machines that run inside the customer's virtual network subnet. These machines form a virtual cluster.

A virtual cluster can host multiple managed instances. If needed, the cluster automatically expands or contracts when the customer changes the number of provisioned instances in the subnet.

Customer applications can connect to managed instances and can query and update databases inside the virtual network, peered virtual network, or network connected by VPN or Azure ExpressRoute. This network must use an endpoint and a private IP address.



Microsoft management and deployment services run outside the virtual network. A managed instance and Microsoft services connect over the endpoints that have public IP addresses. When a managed instance creates an outbound connection, on receiving end Network Address Translation (NAT) makes the connection look like it's coming from this public IP address.

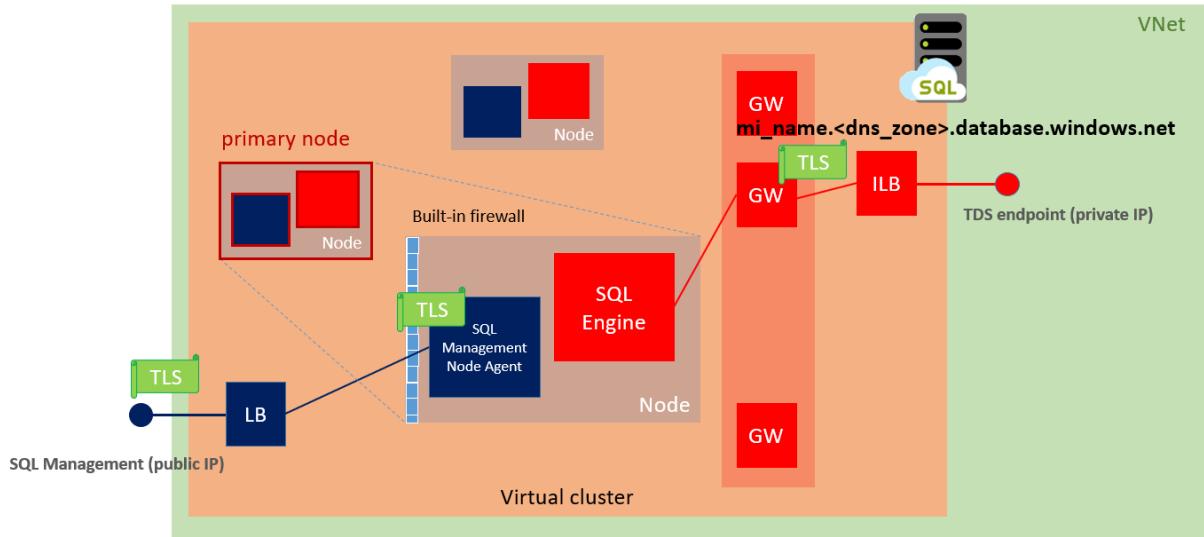
Management traffic flows through the customer's virtual network. That means that elements of the virtual network's infrastructure can harm management traffic by making the instance fail and become unavailable.

### IMPORTANT

To improve customer experience and service availability, Microsoft applies a network intent policy on Azure virtual network infrastructure elements. The policy can affect how the managed instance works. This platform mechanism transparently communicates networking requirements to users. The policy's main goal is to prevent network misconfiguration and to ensure normal managed instance operations. When you delete a managed instance, the network intent policy is also removed.

## Virtual cluster connectivity architecture

Let's take a deeper dive into connectivity architecture for managed instances. The following diagram shows the conceptual layout of the virtual cluster.



Clients connect to a managed instance by using a host name that has the form

`<mi_name>.<dns_zone>.database.windows.net`. This host name resolves to a private IP address although it's registered in a public Domain Name System (DNS) zone and is publicly resolvable. The `zone-id` is automatically generated when you create the cluster. If a newly created cluster hosts a secondary managed instance, it shares its zone ID with the primary cluster. For more information, see [Use auto failover groups to enable transparent and coordinated failover of multiple databases](#).

This private IP address belongs to the managed instance's internal load balancer. The load balancer directs traffic to the managed instance's gateway. Because multiple managed instances can run inside the same cluster, the gateway uses the managed instance's host name to redirect traffic to the correct SQL engine service.

Management and deployment services connect to a managed instance by using a [management endpoint](#) that maps to an external load balancer. Traffic is routed to the nodes only if it's received on a predefined set of ports that only the managed instance's management components use. A built-in firewall on the nodes is set up to allow traffic only from Microsoft IP ranges. Certificates mutually authenticate all communication between management components and the management plane.

## Management endpoint

Microsoft manages the managed instance by using a management endpoint. This endpoint is inside the instance's virtual cluster. The management endpoint is protected by a built-in firewall on the network level. On the application level, it's protected by mutual certificate verification. To find the endpoint's IP address, see [Determine the management endpoint's IP address](#).

When connections start inside the managed instance (as with backups and audit logs), traffic appears to start from the management endpoint's public IP address. You can limit access to public services from a managed instance by setting firewall rules to allow only the managed instance's IP address. For more information, see [Verify the managed instance's built-in firewall](#).

#### NOTE

Traffic that goes to Azure services that are inside the managed instance's region is optimized and for that reason not NATed to managed instance management endpoint public IP address. For that reason if you need to use IP based firewall rules, most commonly for storage, service needs to be in a different region from managed instance.

## Service-aided subnet configuration

To address customer security and manageability requirements Managed Instance is transitioning from manual to service-aided subnet configuration.

With service-aided subnet configuration user is in full control of data (TDS) traffic while Managed Instance takes responsibility to ensure uninterrupted flow of management traffic in order to fulfill SLA.

### Network requirements

Deploy a managed instance in a dedicated subnet inside the virtual network. The subnet must have these characteristics:

- **Dedicated subnet:** The managed instance's subnet can't contain any other cloud service that's associated with it, and it can't be a gateway subnet. The subnet can't contain any resource but the managed instance, and you can't later add other types of resources in the subnet.
- **Subnet delegation:** The managed instance's subnet needs to be delegated to `Microsoft.Sql/managedInstances` resource provider.
- **Network security group (NSG):** A NSG needs to be associated with the managed instance's subnet. You can use an NSG to control access to the managed instance's data endpoint by filtering traffic on port 1433 and ports 11000-11999 when managed instance is configured for redirect connections. Service will automatically add [rules](#) required to allow uninterrupted flow of management traffic.
- **User defined route (UDR) table:** A UDR table needs to be associated with the managed instance's subnet. You can add entries to the route table to route traffic that has on-premises private IP ranges as a destination through the virtual network gateway or virtual network appliance (NVA). Service will automatically add [entries](#) required to allow uninterrupted flow of management traffic.
- **Service endpoints:** Service endpoints could be used to configure virtual network rules on storage accounts that keep backups / audit logs.
- **Sufficient IP addresses:** The managed instance subnet must have at least 16 IP addresses. The recommended minimum is 32 IP addresses. For more information, see [Determine the size of the subnet for managed instances](#). You can deploy managed instances in [the existing network](#) after you configure it to satisfy [the networking requirements for managed instances](#). Otherwise, create a [new network and subnet](#).

#### IMPORTANT

When you create a managed instance, a network intent policy is applied on the subnet to prevent noncompliant changes to networking setup. After the last instance is removed from the subnet, the network intent policy is also removed.

### Mandatory inbound security rules with service-aided subnet configuration

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
management	9000, 9003, 1438, 1440, 1452	TCP	SqlManagement	MI SUBNET	Allow
	9000, 9003	TCP	CorpnetSaw	MI SUBNET	Allow

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
	9000, 9003	TCP	65.55.188.0/24, 167.220.0.0/16, 131.107.0.0/16, 94.245.87.0/24	MI SUBNET	Allow
mi_subnet	Any	Any	MI SUBNET	MI SUBNET	Allow
health_probe	Any	Any	AzureLoadBalancer	MI SUBNET	Allow

#### Mandatory outbound security rules with service-aided subnet configuration

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
management	443, 12000	TCP	MI SUBNET	AzureCloud	Allow
mi_subnet	Any	Any	MI SUBNET	MI SUBNET	Allow

#### User defined routes with service-aided subnet configuration

NAME	ADDRESS PREFIX	NEXT HOP
subnet-to-vnetlocal	MI SUBNET	Virtual network
mi-13-64-11-nexthop-internet	13.64.0.0/11	Internet
mi-13-104-14-nexthop-internet	13.104.0.0/14	Internet
mi-20-34-15-nexthop-internet	20.34.0.0/15	Internet
mi-20-36-14-nexthop-internet	20.36.0.0/14	Internet
mi-20-40-13-nexthop-internet	20.40.0.0/13	Internet
mi-20-128-16-nexthop-internet	20.128.0.0/16	Internet
mi-20-140-15-nexthop-internet	20.140.0.0/15	Internet
mi-20-144-14-nexthop-internet	20.144.0.0/14	Internet
mi-20-150-15-nexthop-internet	20.150.0.0/15	Internet
mi-20-160-12-nexthop-internet	20.160.0.0/12	Internet
mi-20-176-14-nexthop-internet	20.176.0.0/14	Internet
mi-20-180-14-nexthop-internet	20.180.0.0/14	Internet
mi-20-184-13-nexthop-internet	20.184.0.0/13	Internet
mi-40-64-10-nexthop-internet	40.64.0.0/10	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-51-4-15-nexthop-internet	51.4.0.0/15	Internet
mi-51-8-16-nexthop-internet	51.8.0.0/16	Internet
mi-51-10-15-nexthop-internet	51.10.0.0/15	Internet
mi-51-12-15-nexthop-internet	51.12.0.0/15	Internet
mi-51-18-16-nexthop-internet	51.18.0.0/16	Internet
mi-51-51-16-nexthop-internet	51.51.0.0/16	Internet
mi-51-53-16-nexthop-internet	51.53.0.0/16	Internet
mi-51-103-16-nexthop-internet	51.103.0.0/16	Internet
mi-51-104-15-nexthop-internet	51.104.0.0/15	Internet
mi-51-107-16-nexthop-internet	51.107.0.0/16	Internet
mi-51-116-16-nexthop-internet	51.116.0.0/16	Internet
mi-51-120-16-nexthop-internet	51.120.0.0/16	Internet
mi-51-124-16-nexthop-internet	51.124.0.0/16	Internet
mi-51-132-16-nexthop-internet	51.132.0.0/16	Internet
mi-51-136-15-nexthop-internet	51.136.0.0/15	Internet
mi-51-138-16-nexthop-internet	51.138.0.0/16	Internet
mi-51-140-14-nexthop-internet	51.140.0.0/14	Internet
mi-51-144-15-nexthop-internet	51.144.0.0/15	Internet
mi-52-96-12-nexthop-internet	52.96.0.0/12	Internet
mi-52-112-14-nexthop-internet	52.112.0.0/14	Internet
mi-52-125-16-nexthop-internet	52.125.0.0/16	Internet
mi-52-126-15-nexthop-internet	52.126.0.0/15	Internet
mi-52-130-15-nexthop-internet	52.130.0.0/15	Internet
mi-52-132-14-nexthop-internet	52.132.0.0/14	Internet
mi-52-136-13-nexthop-internet	52.136.0.0/13	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-52-145-16-nexthop-internet	52.145.0.0/16	Internet
mi-52-146-15-nexthop-internet	52.146.0.0/15	Internet
mi-52-148-14-nexthop-internet	52.148.0.0/14	Internet
mi-52-152-13-nexthop-internet	52.152.0.0/13	Internet
mi-52-160-11-nexthop-internet	52.160.0.0/11	Internet
mi-52-224-11-nexthop-internet	52.224.0.0/11	Internet
mi-64-4-18-nexthop-internet	64.4.0.0/18	Internet
mi-65-52-14-nexthop-internet	65.52.0.0/14	Internet
mi-66-119-144-20-nexthop-internet	66.119.144.0/20	Internet
mi-70-37-17-nexthop-internet	70.37.0.0/17	Internet
mi-70-37-128-18-nexthop-internet	70.37.128.0/18	Internet
mi-91-190-216-21-nexthop-internet	91.190.216.0/21	Internet
mi-94-245-64-18-nexthop-internet	94.245.64.0/18	Internet
mi-103-9-8-22-nexthop-internet	103.9.8.0/22	Internet
mi-103-25-156-24-nexthop-internet	103.25.156.0/24	Internet
mi-103-25-157-24-nexthop-internet	103.25.157.0/24	Internet
mi-103-25-158-23-nexthop-internet	103.25.158.0/23	Internet
mi-103-36-96-22-nexthop-internet	103.36.96.0/22	Internet
mi-103-255-140-22-nexthop-internet	103.255.140.0/22	Internet
mi-104-40-13-nexthop-internet	104.40.0.0/13	Internet
mi-104-146-15-nexthop-internet	104.146.0.0/15	Internet
mi-104-208-13-nexthop-internet	104.208.0.0/13	Internet
mi-111-221-16-20-nexthop-internet	111.221.16.0/20	Internet
mi-111-221-64-18-nexthop-internet	111.221.64.0/18	Internet
mi-129-75-16-nexthop-internet	129.75.0.0/16	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-131-253-1-24-nexthop-internet	131.253.1.0/24	Internet
mi-131-253-3-24-nexthop-internet	131.253.3.0/24	Internet
mi-131-253-5-24-nexthop-internet	131.253.5.0/24	Internet
mi-131-253-6-24-nexthop-internet	131.253.6.0/24	Internet
mi-131-253-8-24-nexthop-internet	131.253.8.0/24	Internet
mi-131-253-12-22-nexthop-internet	131.253.12.0/22	Internet
mi-131-253-16-23-nexthop-internet	131.253.16.0/23	Internet
mi-131-253-18-24-nexthop-internet	131.253.18.0/24	Internet
mi-131-253-21-24-nexthop-internet	131.253.21.0/24	Internet
mi-131-253-22-23-nexthop-internet	131.253.22.0/23	Internet
mi-131-253-24-21-nexthop-internet	131.253.24.0/21	Internet
mi-131-253-32-20-nexthop-internet	131.253.32.0/20	Internet
mi-131-253-61-24-nexthop-internet	131.253.61.0/24	Internet
mi-131-253-62-23-nexthop-internet	131.253.62.0/23	Internet
mi-131-253-64-18-nexthop-internet	131.253.64.0/18	Internet
mi-131-253-128-17-nexthop-internet	131.253.128.0/17	Internet
mi-132-245-16-nexthop-internet	132.245.0.0/16	Internet
mi-134-170-16-nexthop-internet	134.170.0.0/16	Internet
mi-134-177-16-nexthop-internet	134.177.0.0/16	Internet
mi-137-116-15-nexthop-internet	137.116.0.0/15	Internet
mi-137-135-16-nexthop-internet	137.135.0.0/16	Internet
mi-138-91-16-nexthop-internet	138.91.0.0/16	Internet
mi-138-196-16-nexthop-internet	138.196.0.0/16	Internet
mi-139-217-16-nexthop-internet	139.217.0.0/16	Internet
mi-139-219-16-nexthop-internet	139.219.0.0/16	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-141-251-16-nexthop-internet	141.251.0.0/16	Internet
mi-146-147-16-nexthop-internet	146.147.0.0/16	Internet
mi-147-243-16-nexthop-internet	147.243.0.0/16	Internet
mi-150-171-16-nexthop-internet	150.171.0.0/16	Internet
mi-150-242-48-22-nexthop-internet	150.242.48.0/22	Internet
mi-157-54-15-nexthop-internet	157.54.0.0/15	Internet
mi-157-56-14-nexthop-internet	157.56.0.0/14	Internet
mi-157-60-16-nexthop-internet	157.60.0.0/16	Internet
mi-167-220-16-nexthop-internet	167.220.0.0/16	Internet
mi-168-61-16-nexthop-internet	168.61.0.0/16	Internet
mi-168-62-15-nexthop-internet	168.62.0.0/15	Internet
mi-191-232-13-nexthop-internet	191.232.0.0/13	Internet
mi-192-32-16-nexthop-internet	192.32.0.0/16	Internet
mi-192-48-225-24-nexthop-internet	192.48.225.0/24	Internet
mi-192-84-159-24-nexthop-internet	192.84.159.0/24	Internet
mi-192-84-160-23-nexthop-internet	192.84.160.0/23	Internet
mi-192-100-102-24-nexthop-internet	192.100.102.0/24	Internet
mi-192-100-103-24-nexthop-internet	192.100.103.0/24	Internet
mi-192-197-157-24-nexthop-internet	192.197.157.0/24	Internet
mi-193-149-64-19-nexthop-internet	193.149.64.0/19	Internet
mi-193-221-113-24-nexthop-internet	193.221.113.0/24	Internet
mi-194-69-96-19-nexthop-internet	194.69.96.0/19	Internet
mi-194-110-197-24-nexthop-internet	194.110.197.0/24	Internet
mi-198-105-232-22-nexthop-internet	198.105.232.0/22	Internet
mi-198-200-130-24-nexthop-internet	198.200.130.0/24	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-198-206-164-24-nexthop-internet	198.206.164.0/24	Internet
mi-199-60-28-24-nexthop-internet	199.60.28.0/24	Internet
mi-199-74-210-24-nexthop-internet	199.74.210.0/24	Internet
mi-199-103-90-23-nexthop-internet	199.103.90.0/23	Internet
mi-199-103-122-24-nexthop-internet	199.103.122.0/24	Internet
mi-199-242-32-20-nexthop-internet	199.242.32.0/20	Internet
mi-199-242-48-21-nexthop-internet	199.242.48.0/21	Internet
mi-202-89-224-20-nexthop-internet	202.89.224.0/20	Internet
mi-204-13-120-21-nexthop-internet	204.13.120.0/21	Internet
mi-204-14-180-22-nexthop-internet	204.14.180.0/22	Internet
mi-204-79-135-24-nexthop-internet	204.79.135.0/24	Internet
mi-204-79-179-24-nexthop-internet	204.79.179.0/24	Internet
mi-204-79-181-24-nexthop-internet	204.79.181.0/24	Internet
mi-204-79-188-24-nexthop-internet	204.79.188.0/24	Internet
mi-204-79-195-24-nexthop-internet	204.79.195.0/24	Internet
mi-204-79-196-23-nexthop-internet	204.79.196.0/23	Internet
mi-204-79-252-24-nexthop-internet	204.79.252.0/24	Internet
mi-204-152-18-23-nexthop-internet	204.152.18.0/23	Internet
mi-204-152-140-23-nexthop-internet	204.152.140.0/23	Internet
mi-204-231-192-24-nexthop-internet	204.231.192.0/24	Internet
mi-204-231-194-23-nexthop-internet	204.231.194.0/23	Internet
mi-204-231-197-24-nexthop-internet	204.231.197.0/24	Internet
mi-204-231-198-23-nexthop-internet	204.231.198.0/23	Internet
mi-204-231-200-21-nexthop-internet	204.231.200.0/21	Internet
mi-204-231-208-20-nexthop-internet	204.231.208.0/20	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-204-231-236-24-nexthop-internet	204.231.236.0/24	Internet
mi-205-174-224-20-nexthop-internet	205.174.224.0/20	Internet
mi-206-138-168-21-nexthop-internet	206.138.168.0/21	Internet
mi-206-191-224-19-nexthop-internet	206.191.224.0/19	Internet
mi-207-46-16-nexthop-internet	207.46.0.0/16	Internet
mi-207-68-128-18-nexthop-internet	207.68.128.0/18	Internet
mi-208-68-136-21-nexthop-internet	208.68.136.0/21	Internet
mi-208-76-44-22-nexthop-internet	208.76.44.0/22	Internet
mi-208-84-21-nexthop-internet	208.84.0.0/21	Internet
mi-209-240-192-19-nexthop-internet	209.240.192.0/19	Internet
mi-213-199-128-18-nexthop-internet	213.199.128.0/18	Internet
mi-216-32-180-22-nexthop-internet	216.32.180.0/22	Internet
mi-216-220-208-20-nexthop-internet	216.220.208.0/20	Internet

\* MI SUBNET refers to the IP address range for the subnet in the form 10.x.x.y/ y. You can find this information in the Azure portal, in subnet properties.

In addition, you can add entries to the route table to route traffic that has on-premises private IP ranges as a destination through the virtual network gateway or virtual network appliance (NVA).

If the virtual network includes a custom DNS, the custom DNS server must be able to resolve public dns records. Using additional features like Azure AD Authentication might require resolving additional FQDNs. For more information, see [Set up a custom DNS](#).

#### [Deprecated] Network requirements without service-aided subnet configuration

Deploy a managed instance in a dedicated subnet inside the virtual network. The subnet must have these characteristics:

- **Dedicated subnet:** The managed instance's subnet can't contain any other cloud service that's associated with it, and it can't be a gateway subnet. The subnet can't contain any resource but the managed instance, and you can't later add other types of resources in the subnet.
- **Network security group (NSG):** An NSG that's associated with the virtual network must define [inbound security rules](#) and [outbound security rules](#) before any other rules. You can use an NSG to control access to the managed instance's data endpoint by filtering traffic on port 1433 and ports 11000-11999 when managed instance is configured for redirect connections.
- **User defined route (UDR) table:** A UDR table that's associated with the virtual network must include specific [entries](#).
- **No service endpoints:** No service endpoint should be associated with the managed instance's subnet.

Make sure that the service endpoints option is disabled when you create the virtual network.

- **Sufficient IP addresses:** The managed instance subnet must have at least 16 IP addresses. The recommended minimum is 32 IP addresses. For more information, see [Determine the size of the subnet for managed instances](#). You can deploy managed instances in [the existing network](#) after you configure it to satisfy [the networking requirements for managed instances](#). Otherwise, create a [new network and subnet](#).

#### IMPORTANT

You can't deploy a new managed instance if the destination subnet lacks these characteristics. When you create a managed instance, a network intent policy is applied on the subnet to prevent noncompliant changes to networking setup. After the last instance is removed from the subnet, the network intent policy is also removed.

### Mandatory inbound security rules

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
management	9000, 9003, 1438, 1440, 1452	TCP	Any	MI SUBNET	Allow
mi_subnet	Any	Any	MI SUBNET	MI SUBNET	Allow
health_probe	Any	Any	AzureLoadBalancer	MI SUBNET	Allow

### Mandatory outbound security rules

NAME	PORT	PROTOCOL	SOURCE	DESTINATION	ACTION
management	443, 12000	TCP	MI SUBNET	AzureCloud	Allow
mi_subnet	Any	Any	MI SUBNET	MI SUBNET	Allow

#### IMPORTANT

Ensure there is only one inbound rule for ports 9000, 9003, 1438, 1440, 1452 and one outbound rule for ports 443, 12000. Managed Instance provisioning through Azure Resource Manager deployments will fail if inbound and outbound rules are configured separately for each port. If these ports are in separate rules, the deployment will fail with error code

`VnetSubnetConflictWithIntendedPolicy`

\* MI SUBNET refers to the IP address range for the subnet in the form 10.x.x.x/y. You can find this information in the Azure portal, in subnet properties.

#### IMPORTANT

Although required inbound security rules allow traffic from *any* source on ports 9000, 9003, 1438, 1440, and 1452, these ports are protected by a built-in firewall. For more information, see [Determine the management endpoint address](#).

#### NOTE

If you use transactional replication in a managed instance, and if you use any instance database as a publisher or a distributor, open port 445 (TCP outbound) in the subnet's security rules. This port will allow access to the Azure file share.

## User defined routes

NAME	ADDRESS PREFIX	NEXT HOP
subnet_to_vnetlocal	MI SUBNET	Virtual network
mi-13-64-11-nexthop-internet	13.64.0.0/11	Internet
mi-13-96-13-nexthop-internet	13.96.0.0/13	Internet
mi-13-104-14-nexthop-internet	13.104.0.0/14	Internet
mi-20-8-nexthop-internet	20.0.0.0/8	Internet
mi-23-96-13-nexthop-internet	23.96.0.0/13	Internet
mi-40-64-10-nexthop-internet	40.64.0.0/10	Internet
mi-42-159-16-nexthop-internet	42.159.0.0/16	Internet
mi-51-8-nexthop-internet	51.0.0.0/8	Internet
mi-52-8-nexthop-internet	52.0.0.0/8	Internet
mi-64-4-18-nexthop-internet	64.4.0.0/18	Internet
mi-65-52-14-nexthop-internet	65.52.0.0/14	Internet
mi-66-119-144-20-nexthop-internet	66.119.144.0/20	Internet
mi-70-37-17-nexthop-internet	70.37.0.0/17	Internet
mi-70-37-128-18-nexthop-internet	70.37.128.0/18	Internet
mi-91-190-216-21-nexthop-internet	91.190.216.0/21	Internet
mi-94-245-64-18-nexthop-internet	94.245.64.0/18	Internet
mi-103-9-8-22-nexthop-internet	103.9.8.0/22	Internet
mi-103-25-156-22-nexthop-internet	103.25.156.0/22	Internet
mi-103-36-96-22-nexthop-internet	103.36.96.0/22	Internet
mi-103-255-140-22-nexthop-internet	103.255.140.0/22	Internet
mi-104-40-13-nexthop-internet	104.40.0.0/13	Internet
mi-104-146-15-nexthop-internet	104.146.0.0/15	Internet
mi-104-208-13-nexthop-internet	104.208.0.0/13	Internet
mi-111-221-16-20-nexthop-internet	111.221.16.0/20	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-111-221-64-18-nexthop-internet	111.221.64.0/18	Internet
mi-129-75-16-nexthop-internet	129.75.0.0/16	Internet
mi-131-253-16-nexthop-internet	131.253.0.0/16	Internet
mi-132-245-16-nexthop-internet	132.245.0.0/16	Internet
mi-134-170-16-nexthop-internet	134.170.0.0/16	Internet
mi-134-177-16-nexthop-internet	134.177.0.0/16	Internet
mi-137-116-15-nexthop-internet	137.116.0.0/15	Internet
mi-137-135-16-nexthop-internet	137.135.0.0/16	Internet
mi-138-91-16-nexthop-internet	138.91.0.0/16	Internet
mi-138-196-16-nexthop-internet	138.196.0.0/16	Internet
mi-139-217-16-nexthop-internet	139.217.0.0/16	Internet
mi-139-219-16-nexthop-internet	139.219.0.0/16	Internet
mi-141-251-16-nexthop-internet	141.251.0.0/16	Internet
mi-146-147-16-nexthop-internet	146.147.0.0/16	Internet
mi-147-243-16-nexthop-internet	147.243.0.0/16	Internet
mi-150-171-16-nexthop-internet	150.171.0.0/16	Internet
mi-150-242-48-22-nexthop-internet	150.242.48.0/22	Internet
mi-157-54-15-nexthop-internet	157.54.0.0/15	Internet
mi-157-56-14-nexthop-internet	157.56.0.0/14	Internet
mi-157-60-16-nexthop-internet	157.60.0.0/16	Internet
mi-167-220-16-nexthop-internet	167.220.0.0/16	Internet
mi-168-61-16-nexthop-internet	168.61.0.0/16	Internet
mi-168-62-15-nexthop-internet	168.62.0.0/15	Internet
mi-191-232-13-nexthop-internet	191.232.0.0/13	Internet
mi-192-32-16-nexthop-internet	192.32.0.0/16	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-192-48-225-24-nexthop-internet	192.48.225.0/24	Internet
mi-192-84-159-24-nexthop-internet	192.84.159.0/24	Internet
mi-192-84-160-23-nexthop-internet	192.84.160.0/23	Internet
mi-192-100-102-24-nexthop-internet	192.100.102.0/24	Internet
mi-192-100-103-24-nexthop-internet	192.100.103.0/24	Internet
mi-192-197-157-24-nexthop-internet	192.197.157.0/24	Internet
mi-193-149-64-19-nexthop-internet	193.149.64.0/19	Internet
mi-193-221-113-24-nexthop-internet	193.221.113.0/24	Internet
mi-194-69-96-19-nexthop-internet	194.69.96.0/19	Internet
mi-194-110-197-24-nexthop-internet	194.110.197.0/24	Internet
mi-198-105-232-22-nexthop-internet	198.105.232.0/22	Internet
mi-198-200-130-24-nexthop-internet	198.200.130.0/24	Internet
mi-198-206-164-24-nexthop-internet	198.206.164.0/24	Internet
mi-199-60-28-24-nexthop-internet	199.60.28.0/24	Internet
mi-199-74-210-24-nexthop-internet	199.74.210.0/24	Internet
mi-199-103-90-23-nexthop-internet	199.103.90.0/23	Internet
mi-199-103-122-24-nexthop-internet	199.103.122.0/24	Internet
mi-199-242-32-20-nexthop-internet	199.242.32.0/20	Internet
mi-199-242-48-21-nexthop-internet	199.242.48.0/21	Internet
mi-202-89-224-20-nexthop-internet	202.89.224.0/20	Internet
mi-204-13-120-21-nexthop-internet	204.13.120.0/21	Internet
mi-204-14-180-22-nexthop-internet	204.14.180.0/22	Internet
mi-204-79-135-24-nexthop-internet	204.79.135.0/24	Internet
mi-204-79-179-24-nexthop-internet	204.79.179.0/24	Internet
mi-204-79-181-24-nexthop-internet	204.79.181.0/24	Internet

NAME	ADDRESS PREFIX	NEXT HOP
mi-204-79-188-24-nexthop-internet	204.79.188.0/24	Internet
mi-204-79-195-24-nexthop-internet	204.79.195.0/24	Internet
mi-204-79-196-23-nexthop-internet	204.79.196.0/23	Internet
mi-204-79-252-24-nexthop-internet	204.79.252.0/24	Internet
mi-204-152-18-23-nexthop-internet	204.152.18.0/23	Internet
mi-204-152-140-23-nexthop-internet	204.152.140.0/23	Internet
mi-204-231-192-24-nexthop-internet	204.231.192.0/24	Internet
mi-204-231-194-23-nexthop-internet	204.231.194.0/23	Internet
mi-204-231-197-24-nexthop-internet	204.231.197.0/24	Internet
mi-204-231-198-23-nexthop-internet	204.231.198.0/23	Internet
mi-204-231-200-21-nexthop-internet	204.231.200.0/21	Internet
mi-204-231-208-20-nexthop-internet	204.231.208.0/20	Internet
mi-204-231-236-24-nexthop-internet	204.231.236.0/24	Internet
mi-205-174-224-20-nexthop-internet	205.174.224.0/20	Internet
mi-206-138-168-21-nexthop-internet	206.138.168.0/21	Internet
mi-206-191-224-19-nexthop-internet	206.191.224.0/19	Internet
mi-207-46-16-nexthop-internet	207.46.0.0/16	Internet
mi-207-68-128-18-nexthop-internet	207.68.128.0/18	Internet
mi-208-68-136-21-nexthop-internet	208.68.136.0/21	Internet
mi-208-76-44-22-nexthop-internet	208.76.44.0/22	Internet
mi-208-84-21-nexthop-internet	208.84.0.0/21	Internet
mi-209-240-192-19-nexthop-internet	209.240.192.0/19	Internet
mi-213-199-128-18-nexthop-internet	213.199.128.0/18	Internet
mi-216-32-180-22-nexthop-internet	216.32.180.0/22	Internet
mi-216-220-208-20-nexthop-internet	216.220.208.0/20	Internet

## Next steps

- For an overview, see [SQL Database advanced data security](#).
- Learn how to [set up a new Azure virtual network](#) or an [existing Azure virtual network](#) where you can deploy managed instances.
- [Calculate the size of the subnet](#) where you want to deploy the managed instances.
- Learn how to create a managed instance:
  - From the [Azure portal](#).
  - By using [PowerShell](#).
  - By using [an Azure Resource Manager template](#).
  - By using [an Azure Resource Manager template \(using JumpBox, with SSMS included\)](#).

# Managed instance T-SQL differences, limitations, and known issues

2/18/2020 • 30 minutes to read • [Edit Online](#)

This article summarizes and explains the differences in syntax and behavior between Azure SQL Database managed instance and on-premises SQL Server Database Engine. The managed instance deployment option provides high compatibility with on-premises SQL Server Database Engine. Most of the SQL Server database engine features are supported in a managed instance.

## Easy migration: nearly 100% like SQL Server

<b>Data migration</b> <ul style="list-style-type: none"><li>Native backup/restore</li><li>Configurable DB file layout</li><li>DMS (migrations at scale)</li></ul>	<b>Security</b> <ul style="list-style-type: none"><li>Integrated Auth (Azure AD)</li><li>Encryption (TDE, AE)</li></ul>	<ul style="list-style-type: none"><li>SQL Audit</li><li>Row-Level Security</li><li>Dynamic Data Masking</li></ul>
<b>Programmability</b> <ul style="list-style-type: none"><li>Global temp tables</li><li>Cross-database queries and transactions</li><li>Linked servers</li><li>CLR modules</li></ul>	<b>Operational</b> <ul style="list-style-type: none"><li>DMVs &amp; XEvents</li><li>Query Store</li><li>SQL Agent</li><li>DB Mail (external SMTP)</li></ul>	<b>Scenario enablers</b> <ul style="list-style-type: none"><li>Service Broker</li><li>Change Data Capture</li><li>Transactional Replication</li></ul>

There are some PaaS limitations that are introduced in Managed Instance and some behavior changes compared to SQL Server. The differences are divided into the following categories:

- **Availability** includes the differences in [Always On Availability Groups](#) and [backups](#).
- **Security** includes the differences in [auditing](#), [certificates](#), [credentials](#), [cryptographic providers](#), [logins](#) and [users](#), and the [service key](#) and [service master key](#).
- **Configuration** includes the differences in [buffer pool extension](#), [collation](#), [compatibility levels](#), [database mirroring](#), [database options](#), [SQL Server Agent](#), and [table options](#).
- **Functionalities** include [BULK INSERT/OPENROWSET](#), [CLR](#), [DBCC](#), [distributed transactions](#), [extended events](#), [external libraries](#), [filestream](#) and [FileTable](#), [full-text Semantic Search](#), [linked servers](#), [PolyBase](#), [Replication](#), [RESTORE](#), [Service Broker](#), [stored procedures](#), [functions](#), and [triggers](#).
- **Environment settings** such as VNets and subnet configurations.

Most of these features are architectural constraints and represent service features.

This page also explains [Temporary known issues](#) that are discovered in managed instance, which will be resolved in the future.

## Availability

### Always On Availability Groups

[High availability](#) is built into managed instance and can't be controlled by users. The following statements aren't supported:

- [CREATE ENDPOINT ... FOR DATABASE\\_MIRRORING](#)
- [CREATE AVAILABILITY GROUP](#)
- [ALTER AVAILABILITY GROUP](#)
- [DROP AVAILABILITY GROUP](#)
- The [SET HADR](#) clause of the [ALTER DATABASE](#) statement

## Backup

Managed instances have automatic backups, so users can create full database `COPY_ONLY` backups. Differential, log, and file snapshot backups aren't supported.

- With a managed instance, you can back up an instance database only to an Azure Blob storage account:
  - Only `BACKUP TO URL` is supported.
  - `FILE`, `TAPE`, and backup devices aren't supported.
- Most of the general `WITH` options are supported.
  - `COPY_ONLY` is mandatory.
  - `FILE_SNAPSHOT` isn't supported.
  - Tape options: `REWIND`, `NOREWIND`, `UNLOAD`, and `NOUNLOAD` aren't supported.
  - Log-specific options: `NORECOVERY`, `STANDBY`, and `NO_TRUNCATE` aren't supported.

Limitations:

- With a managed instance, you can back up an instance database to a backup with up to 32 stripes, which is enough for databases up to 4 TB if backup compression is used.
- You can't execute `BACKUP DATABASE ... WITH COPY_ONLY` on a database that's encrypted with service-managed Transparent Data Encryption (TDE). Service-managed TDE forces backups to be encrypted with an internal TDE key. The key can't be exported, so you can't restore the backup. Use automatic backups and point-in-time restore, or use [customer-managed \(BYOK\) TDE](#) instead. You also can disable encryption on the database.
- Manual backups to Azure Blob storage are only supported to [BlockBlobStorage accounts](#).
- The maximum backup stripe size by using the `BACKUP` command in a managed instance is 195 GB, which is the maximum blob size. Increase the number of stripes in the backup command to reduce individual stripe size and stay within this limit.

### TIP

To work around this limitation, when you back up a database from either SQL Server in an on-premises environment or in a virtual machine, you can:

- Back up to `DISK` instead of backing up to `URL`.
- Upload the backup files to Blob storage.
- Restore into the managed instance.

The `Restore` command in a managed instance supports bigger blob sizes in the backup files because a different blob type is used for storage of the uploaded backup files.

For information about backups using T-SQL, see [BACKUP](#).

## Security

### Auditing

The key differences between auditing in databases in Azure SQL Database and databases in SQL Server are:

- With the managed instance deployment option in Azure SQL Database, auditing works at the server level. The `.xe1` log files are stored in Azure Blob storage.
- With the single database and elastic pool deployment options in Azure SQL Database, auditing works at the database level.
- In SQL Server on-premises or virtual machines, auditing works at the server level. Events are stored on file system or Windows event logs.

XEvent auditing in managed instance supports Azure Blob storage targets. File and Windows logs aren't supported.

The key differences in the `CREATE AUDIT` syntax for auditing to Azure Blob storage are:

- A new syntax `TO URL` is provided that you can use to specify the URL of the Azure Blob storage container where the `.xe1` files are placed.
- The syntax `TO FILE` isn't supported because a managed instance can't access Windows file shares.

For more information, see:

- [CREATE SERVER AUDIT](#)
- [ALTER SERVER AUDIT](#)
- [Auditing](#)

## Certificates

A managed instance can't access file shares and Windows folders, so the following constraints apply:

- The `CREATE FROM / BACKUP TO` file isn't supported for certificates.
- The `CREATE / BACKUP` certificate from `FILE / ASSEMBLY` isn't supported. Private key files can't be used.

See [CREATE CERTIFICATE](#) and [BACKUP CERTIFICATE](#).

**Workaround:** Instead of creating backup of certificate and restoring the backup, [get the certificate binary content and private key, store it as .sql file, and create from binary](#):

```
CREATE CERTIFICATE
 FROM BINARY = asn_encoded_certificate
 WITH PRIVATE KEY (<private_key_options>)
```

## Credential

Only Azure Key Vault and `SHARED ACCESS SIGNATURE` identities are supported. Windows users aren't supported.

See [CREATE CREDENTIAL](#) and [ALTER CREDENTIAL](#).

## Cryptographic providers

A managed instance can't access files, so cryptographic providers can't be created:

- `CREATE CRYPTOGRAPHIC PROVIDER` isn't supported. See [CREATE CRYPTOGRAPHIC PROVIDER](#).
- `ALTER CRYPTOGRAPHIC PROVIDER` isn't supported. See [ALTER CRYPTOGRAPHIC PROVIDER](#).

## Logins and users

- SQL logins created by using `FROM CERTIFICATE`, `FROM ASYMMETRIC KEY`, and `FROM SID` are supported. See [CREATE LOGIN](#).
- Azure Active Directory (Azure AD) server principals (logins) created with the [CREATE LOGIN](#) syntax or the [CREATE USER FROM LOGIN \[Azure AD Login\]](#) syntax are supported. These logins are created at the server level.

Managed instance supports Azure AD database principals with the syntax

`CREATE USER [AADUser/AAD group] FROM EXTERNAL PROVIDER`. This feature is also known as Azure AD contained database users.

- Windows logins created with the `CREATE LOGIN ... FROM WINDOWS` syntax aren't supported. Use Azure Active Directory logins and users.
- The Azure AD user who created the instance has **unrestricted admin privileges**.
- Non-administrator Azure AD database-level users can be created by using the `CREATE USER ... FROM EXTERNAL PROVIDER` syntax. See [CREATE USER ... FROM EXTERNAL PROVIDER](#).
- Azure AD server principals (logins) support SQL features within one managed instance only. Features that require cross-instance interaction, no matter whether they're within the same Azure AD tenant or different tenants, aren't supported for Azure AD users. Examples of such features are:
  - SQL transactional replication.
  - Link server.
- Setting an Azure AD login mapped to an Azure AD group as the database owner isn't supported.
- Impersonation of Azure AD server-level principals by using other Azure AD principals is supported, such as the [EXECUTE AS](#) clause. EXECUTE AS limitations are:
  - EXECUTE AS USER isn't supported for Azure AD users when the name differs from the login name. An example is when the user is created through the syntax `CREATE USER [myAadUser] FROM LOGIN [john@contoso.com]` and impersonation is attempted through `EXEC AS USER = myAadUser`. When you create a **USER** from an Azure AD server principal (login), specify the `user_name` as the same `login_name` from **LOGIN**.
  - Only the SQL Server-level principals (logins) that are part of the `sysadmin` role can execute the following operations that target Azure AD principals:
    - `EXECUTE AS USER`
    - `EXECUTE AS LOGIN`
- Database export/import using bacpac files are supported for Azure AD users in managed instance using either [SSMS V18.4 or later](#), or [SQLPackage.exe](#).
  - The following configurations are supported using database bacpac file:
    - Export/import a database between different managed instances within the same Azure AD domain.
    - Export a database from managed instance and import to SQL Database within the same Azure AD domain.
    - Export a database from SQL Database and import to managed instance within the same Azure AD domain.
    - Export a database from managed instance and import to SQL Server (version 2012 or later).
      - In this configuration all Azure AD users are created as SQL database principals (users) without logins. The type of users are listed as SQL (visible as `SQL_USER` in `sys.database_principals`). Their permissions and roles remain in the SQL Server database metadata and can be used for impersonation. However, they cannot be used to access and log in to the SQL Server using their credentials.
- Only the server-level principal login, which is created by the managed instance provisioning process, members of the server roles, such as `securityadmin` or `sysadmin`, or other logins with `ALTER ANY LOGIN` permission at the server level can create Azure AD server principals (logins) in the master database for managed instance.

- If the login is a SQL principal, only logins that are part of the `sysadmin` role can use the `CREATE LOGIN` command to create logins for an Azure AD account.
- The Azure AD login must be a member of an Azure AD within the same directory that's used for Azure SQL Database managed instance.
- Azure AD server principals (logins) are visible in Object Explorer starting with SQL Server Management Studio 18.0 preview 5.
- Overlapping Azure AD server principals (logins) with an Azure AD admin account is allowed. Azure AD server principals (logins) take precedence over the Azure AD admin when you resolve the principal and apply permissions to the managed instance.
- During authentication, the following sequence is applied to resolve the authenticating principal:
  1. If the Azure AD account exists as directly mapped to the Azure AD server principal (login), which is present in `sys.server_principals` as type "E," grant access and apply permissions of the Azure AD server principal (login).
  2. If the Azure AD account is a member of an Azure AD group that's mapped to the Azure AD server principal (login), which is present in `sys.server_principals` as type "X," grant access and apply permissions of the Azure AD group login.
  3. If the Azure AD account is a special portal-configured Azure AD admin for managed instance, which doesn't exist in managed instance system views, apply special fixed permissions of the Azure AD admin for managed instance (legacy mode).
  4. If the Azure AD account exists as directly mapped to an Azure AD user in a database, which is present in `sys.database_principals` as type "E," grant access and apply permissions of the Azure AD database user.
  5. If the Azure AD account is a member of an Azure AD group that's mapped to an Azure AD user in a database, which is present in `sys.database_principals` as type "X," grant access and apply permissions of the Azure AD group login.
  6. If there's an Azure AD login mapped to either an Azure AD user account or an Azure AD group account, which resolves to the user who's authenticating, all permissions from this Azure AD login are applied.

### **Service key and service master key**

- [Master key backup](#) isn't supported (managed by SQL Database service).
- [Master key restore](#) isn't supported (managed by SQL Database service).
- [Service master key backup](#) isn't supported (managed by SQL Database service).
- [Service master key restore](#) isn't supported (managed by SQL Database service).

## **Configuration**

### **Buffer pool extension**

- [Buffer pool extension](#) isn't supported.
- `ALTER SERVER CONFIGURATION SET BUFFER POOL EXTENSION` isn't supported. See [ALTER SERVER CONFIGURATION](#).

### **Collation**

The default instance collation is `SQL_Latin1_General_CI_AS` and can be specified as a creation parameter. See [Collations](#).

### **Compatibility levels**

- Supported compatibility levels are 100, 110, 120, 130, 140 and 150.

- Compatibility levels below 100 aren't supported.
- The default compatibility level for new databases is 140. For restored databases, the compatibility level remains unchanged if it was 100 and above.

See [ALTER DATABASE Compatibility Level](#).

## Database mirroring

Database mirroring isn't supported.

- `ALTER DATABASE SET PARTNER` and `SET WITNESS` options aren't supported.
- `CREATE ENDPOINT ... FOR DATABASE_MIRRORING` isn't supported.

For more information, see [ALTER DATABASE SET PARTNER and SET WITNESS](#) and [CREATE ENDPOINT ... FOR DATABASE\\_MIRRORING](#).

## Database options

- Multiple log files aren't supported.
- In-memory objects aren't supported in the General Purpose service tier.
- There's a limit of 280 files per General Purpose instance, which implies a maximum of 280 files per database. Both data and log files in the General Purpose tier are counted toward this limit. [The Business Critical tier supports 32,767 files per database](#).
- The database can't contain filegroups that contain filestream data. Restore fails if .bak contains `FILESTREAM` data.
- Every file is placed in Azure Blob storage. IO and throughput per file depend on the size of each individual file.

## `CREATE DATABASE` statement

The following limitations apply to `CREATE DATABASE`:

- Files and filegroups can't be defined.
- The `CONTAINMENT` option isn't supported.
- `WITH` options aren't supported.

### TIP

As a workaround, use `ALTER DATABASE` after `CREATE DATABASE` to set database options to add files or to set containment.

- The `FOR ATTACH` option isn't supported.
- The `AS SNAPSHOT OF` option isn't supported.

For more information, see [CREATE DATABASE](#).

## `ALTER DATABASE` statement

Some file properties can't be set or changed:

- A file path can't be specified in the `ALTER DATABASE ADD FILE (FILENAME='path')` T-SQL statement. Remove `FILENAME` from the script because a managed instance automatically places the files.
- A file name can't be changed by using the `ALTER DATABASE` statement.

The following options are set by default and can't be changed:

- `MULTI_USER`

- `ENABLE_BROKER ON`
- `AUTO_CLOSE OFF`

The following options can't be modified:

- `AUTO_CLOSE`
- `AUTOMATIC_TUNING(CREATE_INDEX=ON|OFF)`
- `AUTOMATIC_TUNING(DROP_INDEX=ON|OFF)`
- `DISABLE_BROKER`
- `EMERGENCY`
- `ENABLE_BROKER`
- `FILESTREAM`
- `HADR`
- `NEW_BROKER`
- `OFFLINE`
- `PAGE_VERIFY`
- `PARTNER`
- `READ_ONLY`
- `RECOVERY BULK_LOGGED`
- `RECOVERY_SIMPLE`
- `REMOTE_DATA_ARCHIVE`
- `RESTRICTED_USER`
- `SINGLE_USER`
- `WITNESS`

For more information, see [ALTER DATABASE](#).

## **SQL Server Agent**

- Enabling and disabling SQL Server Agent is currently not supported in managed instance. SQL Agent is always running.
- SQL Server Agent settings are read only. The procedure `sp_set_agent_properties` isn't supported in managed instance.
- Jobs
  - T-SQL job steps are supported.
  - The following replication jobs are supported:
    - Transaction-log reader
    - Snapshot
    - Distributor
  - SSIS job steps are supported.
  - Other types of job steps aren't currently supported:
    - The merge replication job step isn't supported.
    - Queue Reader isn't supported.
    - Command shell isn't yet supported.
  - Managed instances can't access external resources, for example, network shares via robocopy.
  - SQL Server Analysis Services aren't supported.
- Notifications are partially supported.
- Email notification is supported, although it requires that you configure a Database Mail profile. SQL Server Agent can use only one Database Mail profile, and it must be called `AzureManagedInstance_dbmail_profile`.

- Pager isn't supported.
- NetSend isn't supported.
- Alerts aren't yet supported.
- Proxies aren't supported.
- EventLog isn't supported.

The following SQL Agent features currently aren't supported:

- Proxies
- Scheduling jobs on an idle CPU
- Enabling or disabling an Agent
- Alerts

For information about SQL Server Agent, see [SQL Server Agent](#).

## Tables

The following table types aren't supported:

- [FILESTREAM](#)
- [FILETABLE](#)
- [EXTERNAL TABLE](#) (Polybase)
- [MEMORY\\_OPTIMIZED](#) (not supported only in General Purpose tier)

For information about how to create and alter tables, see [CREATE TABLE](#) and [ALTER TABLE](#).

## Functionalities

### Bulk insert / OPENROWSET

A managed instance can't access file shares and Windows folders, so the files must be imported from Azure Blob storage:

- `DATASOURCE` is required in the `BULK INSERT` command while you import files from Azure Blob storage. See [BULK INSERT](#).
- `DATASOURCE` is required in the `OPENROWSET` function when you read the content of a file from Azure Blob storage. See [OPENROWSET](#).
- `OPENROWSET` can be used to read data from other Azure SQL single databases, managed instances or SQL Server instances. Other sources such as Oracle databases or Excel files are not supported.

### CLR

A managed instance can't access file shares and Windows folders, so the following constraints apply:

- Only `CREATE ASSEMBLY FROM BINARY` is supported. See [CREATE ASSEMBLY FROM BINARY](#).
- `CREATE ASSEMBLY FROM FILE` isn't supported. See [CREATE ASSEMBLY FROM FILE](#).
- `ALTER ASSEMBLY` can't reference files. See [ALTER ASSEMBLY](#).

### Database Mail (db\_mail)

- `sp_send_dbmail` cannot send attachments using `@file_attachments` parameter. Local file system and external shares or Azure Blob Storage are not accessible from this procedure.
- See the known issues related to `@query` parameter and authentication.

### DBCC

Undocumented DBCC statements that are enabled in SQL Server aren't supported in managed instances.

- Only a limited number of Global Trace flags are supported. Session-level `Trace flags` aren't supported.

See [Trace flags](#).

- **DBCC TRACEOFF** and **DBCC TRACEON** work with the limited number of global trace-flags.
- **DBCC CHECKDB** with options REPAIR\_ALLOW\_DATA\_LOSS, REPAIR\_FAST, and REPAIR\_REBUILD cannot be used because database cannot be set in **SINGLE\_USER** mode - see [ALTER DATABASE differences](#). Potential database corruptions are handled by Azure support team. Contact Azure support if you are noticing database corruption that should be fixed.

## Distributed transactions

MSDTC and [elastic transactions](#) currently aren't supported in managed instances.

## Extended Events

Some Windows-specific targets for Extended Events (XEvents) aren't supported:

- The **etw\_classic\_sync** target isn't supported. Store **.xel** files in Azure Blob storage. See [etw\\_classic\\_sync target](#).
- The **event\_file** target isn't supported. Store **.xel** files in Azure Blob storage. See [event\\_file target](#).

## External libraries

In-database R and Python, external libraries aren't yet supported. See [SQL Server Machine Learning Services](#).

## Filestream and FileTable

- Filestream data isn't supported.
- The database can't contain filegroups with **FILESTREAM** data.
- **FILETABLE** isn't supported.
- Tables can't have **FILESTREAM** types.
- The following functions aren't supported:
  - **GetPathLocator()**
  - **GET\_FILESTREAM\_TRANSACTION\_CONTEXT()**
  - **PathName()**
  - **GetFileNamespacePat()**
  - **FileTableRootPath()**

For more information, see [FILESTREAM](#) and [FileTables](#).

## Full-text Semantic Search

[Semantic Search](#) isn't supported.

## Linked servers

Linked servers in managed instances support a limited number of targets:

- Supported targets are Managed Instances, Single Databases, and SQL Server instances.
- Linked servers don't support distributed writable transactions (MS DTC).
- Targets that aren't supported are files, Analysis Services, and other RDBMS. Try to use native CSV import from Azure Blob Storage using **BULK INSERT** or **OPENROWSET** as an alternative for file import.

## Operations

- Cross-instance write transactions aren't supported.
- **sp\_dropserver** is supported for dropping a linked server. See [sp\\_dropserver](#).
- The **OPENROWSET** function can be used to execute queries only on SQL Server instances. They can be either managed, on-premises, or in virtual machines. See [OPENROWSET](#).
- The **OPENDATASOURCE** function can be used to execute queries only on SQL Server instances. They can be either managed, on-premises, or in virtual machines. Only the **SQLNCLI**, **SQLNCLI11**, and **SQLOLEDB** values

are supported as a provider. An example is

```
SELECT * FROM OPENDATASOURCE('SQLNCLI', '...').AdventureWorks2012.HumanResources.Employee . See
OPENDATASOURCE.
```

- Linked servers cannot be used to read files (Excel, CSV) from the network shares. Try to use [BULK INSERT](#) or [OPENROWSET](#) that reads CSV files from Azure Blob Storage. Track this requests on [managed instance Feedback item](#)|

## PolyBase

External tables that reference the files in HDFS or Azure Blob storage aren't supported. For information about PolyBase, see [PolyBase](#).

## Replication

- Snapshot and Bi-directional replication types are supported. Merge replication, Peer-to-peer replication, and updatable subscriptions are not supported.
- [Transactional Replication](#) is available for public preview on managed instance with some constraints:
  - All types of replication participants (Publisher, Distributor, Pull Subscriber, and Push Subscriber) can be placed on managed instances, but the publisher and the distributor must be either both in the cloud or both on-premises.
  - Managed instances can communicate with the recent versions of SQL Server. See the [supported versions matrix](#) for more information.
  - Transactional Replication has some [additional networking requirements](#).

For more information about configuring transactional replication, see the following tutorials:

- [Replication between an MI publisher and subscriber](#)
- [Replication between an MI publisher, MI distributor, and SQL Server subscriber](#)

## RESTORE statement

- Supported syntax:
  - `RESTORE DATABASE`
  - `RESTORE FILELISTONLY ONLY`
  - `RESTORE HEADER ONLY`
  - `RESTORE LABELONLY ONLY`
  - `RESTORE VERIFYONLY ONLY`
- Unsupported syntax:
  - `RESTORE LOG ONLY`
  - `RESTORE REWINDONLY ONLY`
- Source:
  - `FROM URL` (Azure Blob storage) is the only supported option.
  - `FROM DISK / TAPE` /backup device isn't supported.
  - Backup sets aren't supported.
- `WITH` options aren't supported, such as no `DIFFERENTIAL` or `STATS`.
- `ASYNC RESTORE` : Restore continues even if the client connection breaks. If your connection is dropped, you can check the `sys.dm_operation_status` view for the status of a restore operation, and for a CREATE and DROP database. See [sys.dm\\_operation\\_status](#).

The following database options are set or overridden and can't be changed later:

- `NEW_BROKER` if the broker isn't enabled in the .bak file.
- `ENABLE_BROKER` if the broker isn't enabled in the .bak file.
- `AUTO_CLOSE=OFF` if a database in the .bak file has `AUTO_CLOSE=ON`.

- `RECOVERY FULL` if a database in the .bak file has `SIMPLE` or `BULK_LOGGED` recovery mode.
- A memory-optimized filegroup is added and called XTP if it wasn't in the source .bak file.
- Any existing memory-optimized filegroup is renamed to XTP.
- `SINGLE_USER` and `RESTRICTED_USER` options are converted to `MULTI_USER`.

Limitations:

- Backups of the corrupted databases might be restored depending on the type of the corruption, but automated backups will not be taken until the corruption is fixed. Make sure that you run `DBCC CHECKDB` on the source instance and use backup `WITH CHECKSUM` in order to prevent this issue.
- Restore of `.BAK` file of a database that contains any limitation described in this document (for example, `FILESTREAM` or `FILETABLE` objects) cannot be restored on Managed Instance.
- `.BAK` files that contain multiple backup sets can't be restored.
- `.BAK` files that contain multiple log files can't be restored.
- Backups that contain databases bigger than 8 TB, active in-memory OLTP objects, or number of files that would exceed 280 files per instance can't be restored on a General Purpose instance.
- Backups that contain databases bigger than 4 TB or in-memory OLTP objects with the total size larger than the size described in [resource limits](#) cannot be restored on Business Critical instance. For information about restore statements, see [RESTORE statements](#).

#### **IMPORTANT**

The same limitations apply to built-in point-in-time restore operation. As an example, General Purpose database greater than 4 TB cannot be restored on Business Critical instance. Business Critical database with In-memory OLTP files or more than 280 files cannot be restored on General Purpose instance.

## **Service broker**

Cross-instance service broker isn't supported:

- `sys.routes` : As a prerequisite, you must select the address from `sys.routes`. The address must be LOCAL on every route. See [sys.routes](#).
- `CREATE ROUTE` : You can't use `CREATE ROUTE` with `ADDRESS` other than `LOCAL`. See [CREATE ROUTE](#).
- `ALTER ROUTE` : You can't use `ALTER ROUTE` with `ADDRESS` other than `LOCAL`. See [ALTER ROUTE](#).

## **Stored procedures, functions, and triggers**

- `NATIVE_COMPILATION` isn't supported in the General Purpose tier.
- The following `sp_configure` options aren't supported:
  - `allow polybase export`
  - `allow updates`
  - `filestream_access_level`
  - `remote data archive`
  - `remote proc trans`
- `sp_execute_external_scripts` isn't supported. See [sp\\_execute\\_external\\_scripts](#).
- `xp_cmdshell` isn't supported. See [xp\\_cmdshell](#).
- `Extended stored procedures` aren't supported, which includes `sp_addextendedproc` and `sp_droptextendedproc`. See [Extended stored procedures](#).
- `sp_attach_db`, `sp_attach_single_file_db`, and `sp_detach_db` aren't supported. See [sp\\_attach\\_db](#), [sp\\_attach\\_single\\_file\\_db](#), and [sp\\_detach\\_db](#).

## **System functions and variables**

The following variables, functions, and views return different results:

- `SERVERPROPERTY('EngineEdition')` returns the value 8. This property uniquely identifies a managed instance. See [SERVERPROPERTY](#).
- `SERVERPROPERTY('InstanceName')` returns NULL because the concept of instance as it exists for SQL Server doesn't apply to a managed instance. See [SERVERPROPERTY\('InstanceName'\)](#).
- `@@SERVERNAMES` returns a full DNS "connectable" name, for example, my-managed-instance.wcus17662feb9ce98.database.windows.net. See [@@SERVERNAMES](#).
- `SYS.SERVERS` returns a full DNS "connectable" name, such as `myinstance.domain.database.windows.net` for the properties "name" and "data\_source." See [SYS.SERVERS](#).
- `@@SERVICENAME` returns NULL because the concept of service as it exists for SQL Server doesn't apply to a managed instance. See [@@SERVICENAME](#).
- `SUSER_ID` is supported. It returns NULL if the Azure AD login isn't in sys.syslogins. See [SUSER\\_ID](#).
- `SUSER_SID` isn't supported. The wrong data is returned, which is a temporary known issue. See [SUSER\\_SID](#).

## Environment constraints

### Subnet

- You cannot place any other resources (for example virtual machines) in the subnet where you have deployed your managed instance. Deploy these resources using a different subnet.
- Subnet must have sufficient number of available [IP addresses](#). Minimum is 16, while recommendation is to have at least 32 IP addresses in the subnet.
- [Service endpoints cannot be associated with the managed instance's subnet](#) Make sure that the service endpoints option is disabled when you create the virtual network.
- The number of vCores and types of instances that you can deploy in a region have some [constraints and limits](#).
- There are some [security rules that must be applied on the subnet](#).

### VNET

- VNet can be deployed using Resource Model - Classic Model for VNet is not supported.
- After a managed instance is created, moving the managed instance or VNet to another resource group or subscription is not supported.
- Some services such as App Service Environments, Logic apps, and managed instances (used for Geo-replication, Transactional replication, or via linked servers) cannot access managed instances in different regions if their VNets are connected using [global peering](#). You can connect to these resources via ExpressRoute or VNet-to-VNet through VNet Gateways.

### TEMPDB

The maximum file size of `tempdb` can't be greater than 24 GB per core on a General Purpose tier. The maximum `tempdb` size on a Business Critical tier is limited by the instance storage size. `Tempdb` log file size is limited to 120 GB on General Purpose tier. Some queries might return an error if they need more than 24 GB per core in `tempdb` or if they produce more than 120 GB of log data.

### MSDB

The following MSDB schemas in managed instance must be owned by their respective predefined roles:

- General roles
  - TargetServersRole
- [Fixed database roles](#)
  - SQLAgentUserRole

- SQLAgentReaderRole
- SQLAgentOperatorRole
- **DatabaseMail roles:**
  - DatabaseMailUserRole
- **Integration services roles:**
  - db\_ssisadmin
  - db\_ssisltduser
  - db\_ssisoperator

#### **IMPORTANT**

Changing the predefined role names, schema names and schema owners by customers will impact the normal operation of the service. Any changes made to these will be reverted back to the predefined values as soon as detected, or at the next service update at the latest to ensure normal service operation.

## **Error logs**

A managed instance places verbose information in error logs. There are many internal system events that are logged in the error log. Use a custom procedure to read error logs that filters out some irrelevant entries. For more information, see [managed instance – sp\\_readmerrorlog](#) or [managed instance extension\(preview\)](#) for Azure Data Studio.

## **Known issues**

### **Limitation of manual failover via portal for failover groups**

**Date:** Jan 2020

If failover group spans across instances in different Azure subscriptions or resource groups, manual failover cannot be initiated from the primary instance in the failover group.

**Workaround:** Initiate failover via portal from the geo-secondary instance.

### **SQL Agent roles need explicit EXECUTE permissions for non-sysadmin logins**

**Date:** Dec 2019

If non-sysadmin logins are added to any of [SQL Agent fixed database roles](#), there exists an issue in which explicit EXECUTE permissions need to be granted to the master stored procedures for these logins to work. If this issue is encountered, the error message "The EXECUTE permission was denied on the object <object\_name> (Microsoft SQL Server, Error: 229)" will be shown.

**Workaround:** Once you add logins to either of SQL Agent fixed database roles: SQLAgentUserRole, SQLAgentReaderRole or SQLAgentOperatorRole, for each of the logins added to these roles execute the below T-SQL script to explicitly grant EXECUTE permissions to the stored procedures listed.

```
USE [master]
GO
CREATE USER [login_name] FOR LOGIN [login_name]
GO
GRANT EXECUTE ON master.dbo.xp_sqlagent_enum_jobs TO [login_name]
GRANT EXECUTE ON master.dbo.xp_sqlagent_is_starting TO [login_name]
GRANT EXECUTE ON master.dbo.xp_sqlagent_notify TO [login_name]
```

### **SQL Agent jobs can be interrupted by Agent process restart**

**Date:** Dec 2019

SQL Agent creates a new session each time job is started, gradually increasing memory consumption. To avoid hitting the internal memory limit which would block execution of scheduled jobs, Agent process will be restarted once its memory consumption reaches threshold. It may result in interrupting execution of jobs running at the moment of restart.

### In-memory OLTP memory limits are not applied

**Date:** Oct 2019

Business Critical service-tier will not correctly apply [max memory limits for memory-optimized objects](#) in some cases. Managed instance may enable workload to use more memory for In-memory OLTP operations, which may affect availability and stability of the instance. In-memory OLTP queries that are reaching the limits might not fail immediately. This issue will be fixed soon. The queries that use more In-memory OLTP memory will fail sooner if they reach the [limits](#).

**Workaround:** [Monitor In-memory OLTP storage usage](#) using [SQL Server Management Studio](#) to ensure that the workload is not using more than available memory. Increase the memory limits that depend on the number of vCores, or optimize your workload to use less memory.

### Wrong error returned while trying to remove a file that is not empty

**Date:** Oct 2019

SQL Server/Managed Instance [don't allow user to drop a file that is not empty](#). If you try to remove a non-empty data file using `ALTER DATABASE REMOVE FILE` statement, the error `Msg 5042 – The file '<file_name>' cannot be removed because it is not empty` will not be immediately returned. Managed Instance will keep trying to drop the file and the operation will fail after 30min with `Internal server error`.

**Workaround:** Remove the content of the file using `DBCC SHRINKFILE (N'<file_name>', EMPTYFILE)` command. If this is the only file in the filegroup you would need to delete data from the table or partition associated to this filegroup before you shrink the file, and optionally load this data into another table/partition.

### Change service tier and create instance operations are blocked by ongoing database restore

**Date:** Sep 2019

Ongoing `RESTORE` statement, Data Migration Service migration process, and built-in point-in time restore will block updating service tier or resize of the existing instance and creating new instances until restore process finishes. Restore process will block these operations on the Managed instances and instance pools in the same subnet where restore process is running. The instances in instance pools are not affected. Create or change service tier operations will not fail or timeout - they will proceed once the restore process is completed or canceled.

**Workaround:** Wait until the restore process finishes, or cancel the restore process if creation or update service-tier operation has higher priority.

### Resource Governor on Business Critical service tier might need to be reconfigured after failover

**Date:** Sep 2019

[Resource Governor](#) feature that enables you to limit the resources assigned to the user workload might incorrectly classify some user workload after failover or user-initiated change of service tier (for example, the change of max vCore or max instance storage size).

**Workaround:** Run `ALTER RESOURCE GOVERNOR RECONFIGURE` periodically or as part of SQL Agent Job that executes the SQL task when the instance starts if you are using [Resource Governor](#).

### Cross-database Service Broker dialogs must be re-initialized after service tier upgrade

**Date:** Aug 2019

Cross-database Service Broker dialogs will stop delivering the messages to the services in other databases after change service tier operation. The messages are **not lost** and they can be found in the sender queue. Any change of vCores or instance storage size in Managed Instance, will cause `service_broke_guid` value in `sys.databases` view to be changed for all databases. Any `DIALOG` created using `BEGIN DIALOG` statement that references Service Brokers in other database will stop delivering messages to the target service.

**Workaround:** Stop any activity that uses cross-database Service Broker dialog conversations before updating service tier and re-initialize them after. If there are remaining messages that are undelivered after service tier change, read the messages from the source queue and resend them to the target queue.

### **Impersonification of Azure AD login types is not supported**

**Date:** July 2019

Impersonation using `EXECUTE AS USER` or `EXECUTE AS LOGIN` of following AAD principals is not supported:

- Aliased AAD users. The following error is returned in this case `15517`.
- AAD logins and users based on AAD applications or service principals. The following errors are returned in this case `15517` and `15406`.

### **@query parameter not supported in sp\_send\_db\_mail**

**Date:** April 2019

The `@query` parameter in the `sp_send_db_mail` procedure doesn't work.

### **Transactional Replication must be reconfigured after geo-failover**

**Date:** Mar 2019

If Transactional Replication is enabled on a database in an auto-failover group, the managed instance administrator must clean up all publications on the old primary and reconfigure them on the new primary after a failover to another region occurs. See [Replication](#) for more details.

### **AAD logins and users are not supported in SSDT**

**Date:** Nov 2019

SQL Server Data Tools don't fully support Azure Active directory logins and users.

### **Temporary database is used during RESTORE operation**

When a database is restoring on Managed Instance, the restore service will first create an empty database with the desired name to allocate the name on the instance. After some time, this database will be dropped and restoring of the actual database will be started. The database that is in *Restoring* state will temporary have a random GUID value instead of name. The temporary name will be changed to the desired name specified in `RESTORE` statement once the restore process completes. In the initial phase, user can access the empty database and even create tables or load data in this database. This temporary database will be dropped when the restore service starts the second phase.

**Workaround:** Do not access the database that you are restoring until you see that restore is completed.

### **TEMPDB structure and content is re-created**

The `tempdb` database is always split into 12 data files and the file structure cannot be changed. The maximum size per file can't be changed, and new files cannot be added to `tempdb`. `Tempdb` is always re-created as an empty database when the instance starts or fails over, and any changes made in `tempdb` will not be preserved.

### **Exceeding storage space with small database files**

`CREATE DATABASE`, `ALTER DATABASE ADD FILE`, and `RESTORE DATABASE` statements might fail because the instance can reach the Azure Storage limit.

Each General Purpose managed instance has up to 35 TB of storage reserved for Azure Premium Disk space.

Each database file is placed on a separate physical disk. Disk sizes can be 128 GB, 256 GB, 512 GB, 1 TB, or 4 TB. Unused space on the disk isn't charged, but the total sum of Azure Premium Disk sizes can't exceed 35 TB. In some cases, a managed instance that doesn't need 8 TB in total might exceed the 35 TB Azure limit on storage size due to internal fragmentation.

For example, a General Purpose managed instance might have one large file that's 1.2 TB in size placed on a 4-TB disk. It also might have 248 files with 1 GB size each that are placed on separate 128-GB disks. In this example:

- The total allocated disk storage size is  $1 \times 4\text{ TB} + 248 \times 128\text{ GB} = 35\text{ TB}$ .
- The total reserved space for databases on the instance is  $1 \times 1.2\text{ TB} + 248 \times 1\text{ GB} = 1.4\text{ TB}$ .

This example illustrates that under certain circumstances, due to a specific distribution of files, a managed instance might reach the 35-TB limit that's reserved for an attached Azure Premium Disk when you might not expect it to.

In this example, existing databases continue to work and can grow without any problem as long as new files aren't added. New databases can't be created or restored because there isn't enough space for new disk drives, even if the total size of all databases doesn't reach the instance size limit. The error that's returned in that case isn't clear.

You can [identify the number of remaining files](#) by using system views. If you reach this limit, try to [empty and delete some of the smaller files by using the DBCC SHRINKFILE statement](#) or switch to the [Business Critical tier, which doesn't have this limit](#).

#### **GUID values shown instead of database names**

Several system views, performance counters, error messages, XEvents, and error log entries display GUID database identifiers instead of the actual database names. Don't rely on these GUID identifiers because they're replaced with actual database names in the future.

#### **Error logs aren't persisted**

Error logs that are available in managed instance aren't persisted, and their size isn't included in the maximum storage limit. Error logs might be automatically erased if failover occurs. There might be gaps in the error log history because Managed Instance was moved several times on several virtual machines.

#### **Transaction scope on two databases within the same instance isn't supported**

The `TransactionScope` class in .NET doesn't work if two queries are sent to two databases within the same instance under the same transaction scope:

```

using (var scope = new TransactionScope())
{
 using (var conn1 = new
SqlConnection("Server=quickstartbmi.neu15011648751ff.database.windows.net;Database=b;User
ID=myuser;Password=mypassword;Encrypt=true"))
 {
 conn1.Open();
 SqlCommand cmd1 = conn1.CreateCommand();
 cmd1.CommandText = string.Format("insert into T1 values(1)");
 cmd1.ExecuteNonQuery();
 }

 using (var conn2 = new
SqlConnection("Server=quickstartbmi.neu15011648751ff.database.windows.net;Database=b;User
ID=myuser;Password=mypassword;Encrypt=true"))
 {
 conn2.Open();
 var cmd2 = conn2.CreateCommand();
 cmd2.CommandText = string.Format("insert into b.dbo.T2 values(2)"); cmd2.ExecuteNonQuery();
 }

 scope.Complete();
}

```

Although this code works with data within the same instance, it required MSDTC.

**Workaround:** Use [SqlConnection.ChangeDatabase\(String\)](#) to use another database in a connection context instead of using two connections.

#### CLR modules and linked servers sometimes can't reference a local IP address

CLR modules placed in a managed instance and linked servers or distributed queries that reference a current instance sometimes can't resolve the IP of a local instance. This error is a transient issue.

**Workaround:** Use context connections in a CLR module if possible.

## Next steps

- For more information about managed instances, see [What is a managed instance?](#)
- For a features and comparison list, see [Azure SQL Database feature comparison](#).
- For a quickstart that shows you how to create a new managed instance, see [Create a managed instance](#).

# How to use a managed instance in Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

In this article you can find various guides, scripts, and explanation that can help you to manage and configure your managed instance.

## Migration

- [Migrate to a managed instance](#) – Learn about the recommended migration process and tools for migration to a managed instance.
- [Migrate TDE cert to a managed instance](#) – If your SQL Server database is protected with transparent data encryption (TDE), you would need to migrate certificate that a managed instance can use to decrypt the backup that you want to restore in Azure.

## Network configuration

- [Determine size of a managed instance subnet](#) – Managed instance is placed in dedicated subnet that cannot be resized once you add the resources inside. Therefore, you would need to calculate what IP range of addresses would be required for the subnet depending on the number and types of instances that you want to deploy in the subnet.
- [Create new VNet and subnet for a managed instance](#) – Azure VNet and subnet where you want to deploy your managed instances must be configured according to the [network requirements described here](#). In this guide you can find the easiest way to create your new VNet and subnet properly configured for managed instances.
- [Configure existing VNet and subnet for a managed instance](#) – if you want to configure your existing VNet and subnet to deploy managed instances inside, here you can find the script that checks the [network requirements](#) and make configurations your subnet according to the requirements.
- [Configure custom DNS](#) – you need to configure custom DNS if you want to access external resources on the custom domains from your managed instance via linked server or db mail profiles.
- [Sync network configuration](#) – It might happen that although you [integrated your app with an Azure Virtual Network](#), you can't establish connection to a managed instance. One thing you can try is to refresh networking configuration for your service plan.
- [Find management endpoint IP address](#) – Managed instance uses public endpoint for management-purposes. You can determine IP address of the management endpoint using the script described here.
- [Verify built-in firewall protection](#) – Managed instance is protected with built-in firewall that allows the traffic only on necessary ports. You can check and verify the built-in firewall rules using the script described in this guide.
- [Connect applications](#) – Managed instance is placed in your own private Azure VNet with private IP address. Learn about different patterns for connecting the applications to your managed instance.

## Feature configuration

- [Transactional replication](#) enables you to replicate your data between managed instances, or from on-premises SQL Server to a managed instance, and vice versa. Find more information how to use and configure transaction replication in this guide.
- [Configure threat detection](#) – [threat detection](#) is a built-in Azure SQL Database feature that detects various potential attacks such as SQL Injection or access from suspicious locations. In this guide you can learn how to

enable and configure [threat detection](#) for a managed instance.

## Next steps

- Learn more about [How-to guides for single databases](#)

# SQL Server instance migration to Azure SQL Database managed instance

11/7/2019 • 17 minutes to read • [Edit Online](#)

In this article, you learn about the methods for migrating a SQL Server 2005 or later version instance to [Azure SQL Database managed instance](#). For information on migrating to a single database or elastic pool, see [Migrate to a single or pooled database](#). For migration information about migrating from other platforms, see [Azure Database Migration Guide](#).

## NOTE

If you want to quickly start and try Managed Instance, you might want to go to [Quick-start guide](#) instead of this page.

At a high level, the database migration process looks like:



- [Assess managed instance compatibility](#) where you should ensure that there are no blocking issues that can prevent your migrations.
  - This step also includes creation of [performance baseline](#) to determine resource usage on your source SQL Server instance. This step is needed if you want to deploy properly sized Managed Instance and verify that performances after migration are not affected.
- [Choose app connectivity options](#)
- [Deploy to an optimally sized managed instance](#) where you will choose technical characteristics (number of vCores, amount of memory) and performance tier (Business Critical, General Purpose) of your Managed Instance.
- [Select migration method and migrate](#) where you migrate your databases using offline migration (native backup/restore, database import/export) or online migration (Data Migration Service, Transactional Replication).
- [Monitor applications](#) to ensure that you have expected performance.

## NOTE

To migrate an individual database into either a single database or elastic pool, see [Migrate a SQL Server database to Azure SQL Database](#).

## Assess managed instance compatibility

First, determine whether managed instance is compatible with the database requirements of your application. The managed instance deployment option is designed to provide easy lift and shift migration for the majority of existing applications that use SQL Server on-premises or on virtual machines. However, you may sometimes require features or capabilities that are not yet supported and the cost of implementing a workaround are too high.

Use [Data Migration Assistant \(DMA\)](#) to detect potential compatibility issues impacting database functionality on Azure SQL Database. DMA does not yet support managed instance as migration destination, but it is recommended to run assessment against Azure SQL Database and carefully review list of reported feature parity and compatibility issues against product documentation. See [Azure SQL Database features](#) to check if there are some reported blocking issues that are not blockers in managed instance, because most of the blocking issues preventing a migration to Azure SQL Database have been removed with managed instance. For instance, features like cross-database queries, cross-database transactions within the same instance, linked server to other SQL sources, CLR, global temp tables, instance level views, Service Broker and the like are available in managed instances.

If there are some reported blocking issues that are not removed with the managed instance deployment option, you might need to consider an alternative option, such as [SQL Server on Azure virtual machines](#). Here are some examples:

- If you require direct access to the operating system or file system, for instance to install third party or custom agents on the same virtual machine with SQL Server.
- If you have strict dependency on features that are still not supported, such as FileStream / FileTable, PolyBase, and cross-instance transactions.
- If absolutely need to stay at a specific version of SQL Server (2012, for instance).
- If your compute requirements are much lower than managed instance offers (one vCore, for instance) and database consolidation is not acceptable option.

If you have resolved all identified migration blockers and continuing the migration to Managed Instance, note that some of the changes might affect performance of your workload:

- Mandatory full recovery model and regular automated backup schedule might impact performance of your workload or maintenance/ETL actions if you have periodically used simple/bulk-logged model or stopped backups on demand.
- Different server or database level configurations such as trace flags or compatibility levels
- New features that you are using such as Transparent Database Encryption (TDE) or auto-failover groups might impact CPU and IO usage.

Managed Instance guarantee 99.99% availability even in the critical scenarios, so overhead caused by these features cannot be disabled. For more information, see [the root causes that might cause different performance on SQL Server and Managed Instance](#).

### Create performance baseline

If you need to compare the performance of your workload on Managed Instance with your original workload running on SQL Server, you would need to create a performance baseline that will be used for comparison.

Performance baseline is a set of parameters such as average/max CPU usage, average/max disk IO latency, throughput, IOPS, average/max page life expectancy, average max size of tempdb. You would like to have similar or even better parameters after migration, so it is important to measure and record the baseline values for these parameters. In addition to system parameters, you would need to select a set of the representative queries or the most important queries in your workload and measure min/average/max duration, CPU usage for the selected queries. These values would enable you to compare performance of workload running on Managed Instance to the original values on your source SQL Server.

Some of the parameters that you would need to measure on your SQL Server instance are:

- [Monitor CPU usage on your SQL Server instance](#) and record the average and peak CPU usage.
- [Monitor memory usage on your SQL Server instance](#) and determine the amount of memory used by different components such as buffer pool, plan cache, column-store pool, [In-memory OLTP](#), etc. In addition, you should find average and peak values of Page Life Expectancy memory performance counter.
- Monitor disk IO usage on the source SQL Server instance using [sys.dm\\_io\\_virtual\\_file\\_stats](#) view or

performance counters.

- Monitor workload and query performance on your SQL Server instance by examining Dynamic Management Views or Query Store if you are migrating from SQL Server 2016+ version. Identify average duration and CPU usage of the most important queries in your workload to compare them with the queries that are running on the Managed Instance.

#### NOTE

If you notice any issue with your workload on SQL Server such as high CPU usage, constant memory pressure, tempdb or parametrization issues, you should try to resolve them on your source SQL Server instance before taking the baseline and migration. Migrating known issues to any new system might cause unexpected results and invalidate any performance comparison.

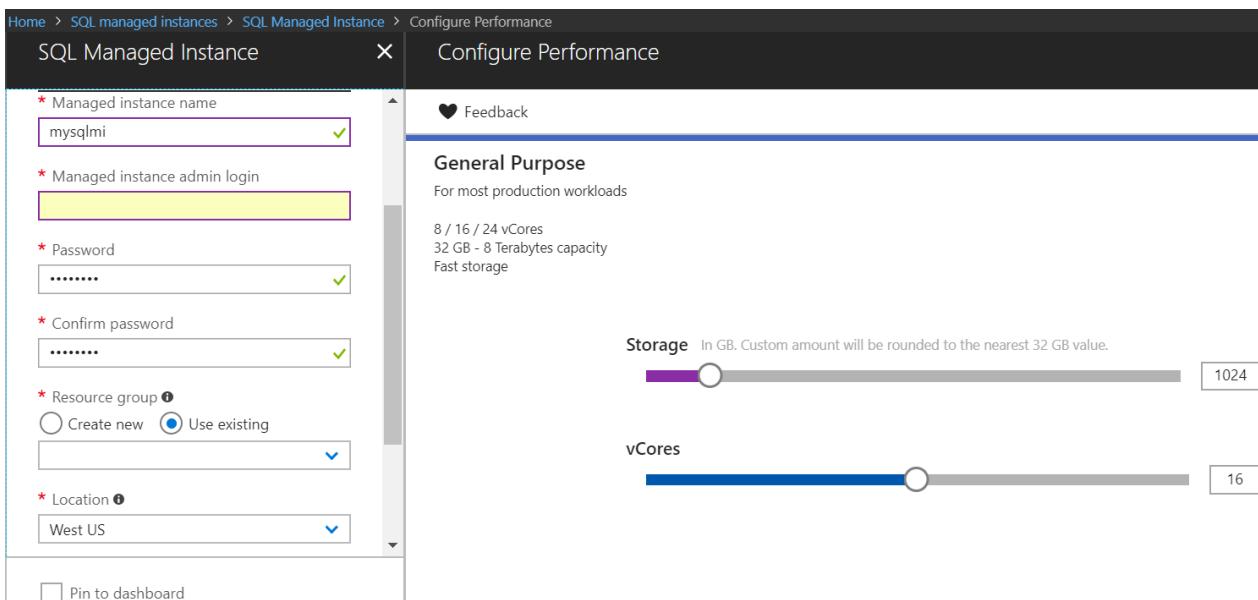
As an outcome of this activity you should have documented average and peak values for CPU, memory, and IO usage on your source system, as well as average and max duration and CPU usage of the dominant and the most critical queries in your workload. You should use these values later to compare performance of your workload on Managed Instance with the baseline performance of the workload on the source SQL Server.

## Deploy to an optimally-sized managed instance

Managed instance is tailored for on-premises workloads that are planning to move to the cloud. It introduces a [new purchasing model](#) that provides greater flexibility in selecting the right level of resources for your workloads. In the on-premises world, you are probably accustomed to sizing these workloads by using physical cores and IO bandwidth. The purchasing model for managed instance is based upon virtual cores, or "vCores," with additional storage and IO available separately. The vCore model is a simpler way to understand your compute requirements in the cloud versus what you use on-premises today. This new model enables you to right-size your destination environment in the cloud. Some general guidelines that might help you to choose the right service tier and characteristics are described here:

- Based on the baseline CPU usage you can provision a Managed Instance that matches the number of cores that you are using on SQL Server, having in mind that CPU characteristics might need to be scaled to match [VM characteristics where Managed Instance is installed](#).
- Based on the baseline memory usage choose [the service tier that has matching memory](#). The amount of memory cannot be directly chosen so you would need to select the Managed Instance with the amount of vCores that has matching memory (for example 5.1 GB/vCore in Gen5).
- Based on the baseline IO latency of the file subsystem choose between General Purpose (latency greater than 5ms) and Business Critical service tiers (latency less than 3 ms).
- Based on baseline throughput pre-allocate the size of data or log files to get expected IO performance.

You can choose compute and storage resources at deployment time and then change it afterwards without introducing downtime for your application using the [Azure portal](#):



To learn how to create the VNet infrastructure and a managed instance, see [Create a managed instance](#).

#### IMPORTANT

It is important to keep your destination VNet and subnet always in accordance with [managed instance VNet requirements](#). Any incompatibility can prevent you from creating new instances or using those that you already created. Learn more about [creating new](#) and [configuring existing](#) networks.

## Select migration method and migrate

The managed instance deployment option targets user scenarios requiring mass database migration from on-premises or IaaS database implementations. They are optimal choice when you need to lift and shift the back end of the applications that regularly use instance level and / or cross-database functionalities. If this is your scenario, you can move an entire instance to a corresponding environment in Azure without the need to re-architect your applications.

To move SQL instances, you need to plan carefully:

- The migration of all databases that need to be collocated (ones running on the same instance)
- The migration of instance-level objects that your application depends on, including logins, credentials, SQL Agent Jobs and Operators, and server level triggers.

Managed instance is a managed service that allows you to delegate some of the regular DBA activities to the platform as they are built in. Therefore, some instance level data does not need to be migrated, such as maintenance jobs for regular backups or Always On configuration, as [high availability](#) is built in.

Managed instance supports the following database migration options (currently these are the only supported migration methods):

- Azure Database Migration Service - migration with near-zero downtime,
- Native `RESTORE DATABASE FROM URL` - uses native backups from SQL Server and requires some downtime.

### Azure Database Migration Service

The [Azure Database Migration Service \(DMS\)](#) is a fully managed service designed to enable seamless migrations from multiple database sources to Azure Data platforms with minimal downtime. This service streamlines the tasks required to move existing third party and SQL Server databases to Azure. Deployment options at public preview include databases in Azure SQL Database and SQL Server databases in an Azure Virtual Machine. DMS is the recommended method of migration for your enterprise workloads.

If you use SQL Server Integration Services (SSIS) on your SQL Server on-premises, DMS does not yet support migrating SSIS catalog (SSISDB) that stores SSIS packages, but you can provision Azure-SSIS Integration Runtime (IR) in Azure Data Factory (ADF) that will create a new SSISDB in a managed instance and then you can redeploy your packages to it, see [Create Azure-SSIS IR in ADF](#).

To learn more about this scenario and configuration steps for DMS, see [Migrate your on-premises database to managed instance using DMS](#).

### Native RESTORE from URL

RESTORE of native backups (.bak files) taken from SQL Server on-premises or [SQL Server on Virtual Machines](#), available on [Azure Storage](#), is one of key capabilities of the managed instance deployment option that enables quick and easy offline database migration.

The following diagram provides a high-level overview of the process:



The following table provides more information regarding the methods you can use depending on source SQL Server version you are running:

STEP	SQL ENGINE AND VERSION	BACKUP / RESTORE METHOD
Put backup to Azure Storage	Prior SQL 2012 SP1 CU2	Upload .bak file directly to Azure storage
	2012 SP1 CU2 - 2016	Direct backup using deprecated <a href="#">WITH CREDENTIAL</a> syntax
	2016 and above	Direct backup using <a href="#">WITH SAS CREDENTIAL</a>
Restore from Azure storage to managed instance	<a href="#">RESTORE FROM URL with SAS CREDENTIAL</a>	

#### IMPORTANT

- When migrating a database protected by [Transparent Data Encryption](#) to a managed instance using native restore option, the corresponding certificate from the on-premises or IaaS SQL Server needs to be migrated before database restore. For detailed steps, see [Migrate TDE cert to managed instance](#)
- Restore of system databases is not supported. To migrate instance level objects (stored in master or msdb databases), we recommend to script them out and run T-SQL scripts on the destination instance.

For a quickstart showing how to restore a database backup to a managed instance using a SAS credential, see [Restore from backup to a managed instance](#).

## Monitor applications

Once you have completed the migration to Managed Instance, you should track the application behavior and

performance of your workload. This process includes the following activities:

- Compare performance of the workload running on the Managed Instance with the performance baseline that you created on the source SQL Server.
- Continuously monitor performance of your workload to identify potential issues and improvement.

### Compare performance with the baseline

The first activity that you would need to take immediately after successful migration is to compare the performance of the workload with the baseline workload performance. The goal of this activity is to confirm that the workload performance on your Managed Instance meets your needs.

Database migration to Managed Instance keeps database settings and its original compatibility level in majority of cases. The original settings are preserved where possible in order to reduce risk of some performance degradations compared to your source SQL Server. If the compatibility level of a user database was 100 or higher before the migration, it remains the same after migration. If the compatibility level of a user database was 90 before migration, in the upgraded database, the compatibility level is set to 100, which is the lowest supported compatibility level in managed instance. Compatibility level of system databases is 140. Since migration to Managed Instance is actually migrating to the latest version of SQL Server Database Engine, you should be aware that you need to re-test performance of your workload to avoid some surprising performance issues.

As a prerequisite, make sure that you have completed the following activities:

- Align your settings on Managed Instance with the settings from the source SQL Server instance by investigating various instance, database, temdb settings, and configurations. Make sure that you have not changed settings like compatibility levels or encryption before you run the first performance comparison, or accept the risk that some of the new features that you enabled might affect some queries. To reduce migration risks, change the database compatibility level only after performance monitoring.
- Implement [storage best practice guidelines for General Purpose](#) such as pre-allocating the size of the files to get the better performance.
- Learn about the [key environment differences that might cause the performance differences between Managed Instance and SQL Server](#) and identify the risks that might affect the performance.
- Make sure that you keep enabled Query Store and Automatic tuning on your Managed Instance. These features enable you to measure workload performance and automatically fix the potential performance issues. Learn how to use Query Store as an optimal tool for getting information about workload performance before and after database compatibility level change, as explained in [Keep performance stability during the upgrade to newer SQL Server version](#). Once you have prepared the environment that is comparable as much as possible with your on-premises environment, you can start running your workload and measure performance. Measurement process should include the same parameters that you measured [while you create baseline performance of your workload measures on the source SQL Server](#). As a result, you should compare performance parameters with the baseline and identify critical differences.

#### NOTE

In many cases, you would not be able to get exactly matching performance on Managed Instance and SQL Server. Managed Instance is a SQL Server database engine but infrastructure and High-availability configuration on Managed Instance may introduce some difference. You might expect that some queries would be faster while some other might be slower. The goal of comparison is to verify that workload performance in Managed Instance matches the performance on SQL Server (in average), and identify are there any critical queries with the performance that don't match your original performance.

The outcome of the performance comparison might be:

- Workload performance on Managed Instance is aligned or better than the workload performance on SQL Server. In this case you have successfully confirmed that migration is successful.

- Majority of the performance parameters and the queries in the workload work fine, with some exceptions with degraded performance. In this case, you would need to identify the differences and their importance. If there are some important queries with degraded performance, you should investigate are the underlying SQL plans changed or the queries are hitting some resource limits. Mitigation in this case could be to apply some hints on the critical queries (for example changed compatibility level, legacy cardinality estimator) either directly or using plan guides, rebuild or create statistics and indexes that might affect the plans.
- Most of the queries are slower on Managed Instance compared to your source SQL Server. In this case try to identify the root causes of the difference such as [reaching some resource limit](#) like IO limits, memory limit, instance log rate limit, etc. If there are no resource limits that can cause the difference, try to change compatibility level of the database or change database settings like legacy cardinality estimation and re-start the test. Review the recommendations provided by Managed Instance or Query Store views to identify the queries that regressed performance.

#### **IMPORTANT**

Managed Instance has built-in automatic plan correction feature that is enabled by default. This feature ensures that queries that worked fine in the past would not degrade in the future. Make sure that this feature is enabled and that you have executed the workload long enough with the old settings before you change new settings in order to enable Managed Instance to learn about the baseline performance and plans.

Make the change of the parameters or upgrade service tiers to converge to the optimal configuration until you get the workload performance that fits your needs.

#### **Monitor performance**

Managed Instance provides a lot of advanced tools for monitoring and troubleshooting, and you should use them to monitor performance on your instance. Some of the parameters that you would need to monitor are:

- CPU usage on the instance to determine does the number of vCores that you provisioned is the right match for your workload.
- Page-life expectancy on your Managed Instance to determine [do you need additional memory](#).
- Wait statistics like `INSTANCE_LOG_GOVERNOR` or `PAGEIOLATCH` that will tell do you have storage IO issues, especially on General Purpose tier where you might need to pre-allocate files to get better IO performance.

## Leverage advanced PaaS features

Once you are on a fully managed platform and you have verified that workload performances are matching your SQL Server workload performance, take advantages that are provided automatically as part of the SQL Database service.

Even if you don't make some changes in managed instance during the migration, there are high chances that you would turn on some of the new features while you are operating your instance to take an advantage of the latest database engine improvements. Some changes are only enabled once the [database compatibility level has been changed](#).

For instance, you don't have to create backups on managed instance - the service performs backups for you automatically. You no longer must worry about scheduling, taking, and managing backups. Managed instance provides you the ability to restore to any point in time within this retention period using [Point in Time Recovery \(PITR\)](#). Additionally, you do not need to worry about setting up high availability as [high availability](#) is built in.

To strengthen security, consider using [Azure Active Directory Authentication, Auditing, threat detection, row-level security](#), and [dynamic data masking](#) ).

In addition to advanced management and security features, Managed Instance provides a set of advanced tools that can help you to [monitor and tune your workload](#). [Azure SQL analytics](#) enables you to monitor a large set of

Managed Instances and centralize monitoring of a large number of instances and databases. [Automatic tuning](#) in Managed Instance continuously monitor performance of your SQL plan execution statistics and automatically fix the identified performance issues.

## Next steps

- For information about managed instances, see [What is a managed instance?](#).
- For a tutorial that includes a restore from backup, see [Create a managed instance](#).
- For tutorial showing migration using DMS, see [Migrate your on-premises database to managed instance using DMS](#).

# Time zones in Azure SQL Database Managed Instance

11/7/2019 • 7 minutes to read • [Edit Online](#)

Coordinated Universal Time (UTC) is the recommended time zone for the data tier of cloud solutions. Azure SQL Database Managed Instance also offers a choice of time zones to meet the needs of existing applications that store date and time values and call date and time functions with an implicit context of a specific time zone.

T-SQL functions like [GETDATE\(\)](#) or CLR code observe the time zone set on the instance level. SQL Server Agent jobs also follow schedules according to the time zone of the instance.

## NOTE

Managed Instance is the only deployment option of Azure SQL Database that supports time zone setting. Other deployment options always follow UTC. Use [AT TIME ZONE](#) in single and pooled SQL databases if you need to interpret date and time information in a non-UTC time zone.

## Supported time zones

A set of supported time zones is inherited from the underlying operating system of the managed instance. It's regularly updated to get new time zone definitions and reflect changes to the existing ones.

[Daylight saving time/time zone changes policy](#) guarantees historical accuracy from 2010 forward.

A list with names of the supported time zones is exposed through the [sys.time\\_zone\\_info](#) system view.

## Set a time zone

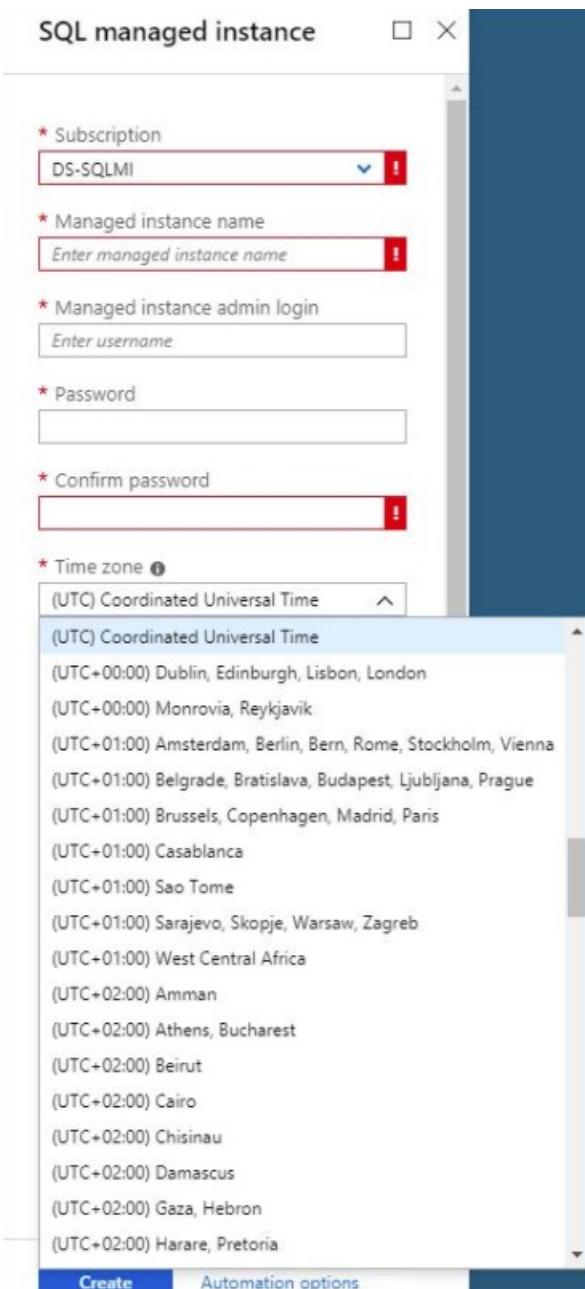
A time zone of a managed instance can be set during instance creation only. The default time zone is UTC.

## NOTE

The time zone of an existing managed instance can't be changed.

## Set the time zone through the Azure portal

When you enter parameters for a new instance, select a time zone from the list of supported time zones.



## Azure Resource Manager template

Specify the `timeZoneId` property in your [Resource Manager template](#) to set the time zone during instance creation.

```
"properties": {
 "administratorLogin": "[parameters('user')]",
 "administratorLoginPassword": "[parameters('pwd')]",
 "subnetId": "[parameters('subnetId')]",
 "storageSizeInGB": 256,
 "vCores": 8,
 "licenseType": "LicenseIncluded",
 "hardwareFamily": "Gen5",
 "collation": "Serbian_Cyrillic_100_CS_AS",
 "timeZoneId": "Central European Standard Time"
},
```

A list of supported values for the `timeZoneId` property is at the end of this article.

If not specified, the time zone is set to UTC.

## Check the time zone of an instance

The [CURRENT\\_TIMEZONE](#) function returns a display name of the time zone of the instance.

## Cross-feature considerations

### Restore and import

You can restore a backup file or import data to a managed instance from an instance or a server with different time zone settings. Make sure to do so with caution. Analyze the application behavior and the results of the queries and reports, just like when you transfer data between two SQL Server instances with different time zone settings.

### Point-in-time restore

When you perform a point-in-time restore, the time to restore to is interpreted as UTC time. This way any ambiguities due to daylight saving time and its potential changes are avoided.

### Auto-failover groups

Using the same time zone across a primary and secondary instance in a failover group isn't enforced, but we strongly recommend it.

#### WARNING

We strongly recommend that you use the same time zone for the primary and secondary instance in a failover group. Because of certain rare use cases keeping the same time zone across primary and secondary instances isn't enforced. It's important to understand that in the case of manual or automatic failover, the secondary instance will retain its original time zone.

## Limitations

- The time zone of the existing managed instance can't be changed.
- External processes launched from the SQL Server Agent jobs don't observe the time zone of the instance.

## List of supported time zones

TIME ZONE ID	TIME ZONE DISPLAY NAME
Dateline Standard Time	(UTC-12:00) International Date Line West
UTC-11	(UTC-11:00) Coordinated Universal Time-11
Aleutian Standard Time	(UTC-10:00) Aleutian Islands
Hawaiian Standard Time	(UTC-10:00) Hawaii
Marquesas Standard Time	(UTC-09:30) Marquesas Islands
Alaskan Standard Time	(UTC-09:00) Alaska
UTC-09	(UTC-09:00) Coordinated Universal Time-09
Pacific Standard Time (Mexico)	(UTC-08:00) Baja California
UTC-08	(UTC-08:00) Coordinated Universal Time-08
Pacific Standard Time	(UTC-08:00) Pacific Time (US & Canada)

TIME ZONE ID	TIME ZONE DISPLAY NAME
US Mountain Standard Time	(UTC-07:00) Arizona
Mountain Standard Time (Mexico)	(UTC-07:00) Chihuahua, La Paz, Mazatlan
Mountain Standard Time	(UTC-07:00) Mountain Time (US & Canada)
Central America Standard Time	(UTC-06:00) Central America
Central Standard Time	(UTC-06:00) Central Time (US & Canada)
Easter Island Standard Time	(UTC-06:00) Easter Island
Central Standard Time (Mexico)	(UTC-06:00) Guadalajara, Mexico City, Monterrey
Canada Central Standard Time	(UTC-06:00) Saskatchewan
SA Pacific Standard Time	(UTC-05:00) Bogota, Lima, Quito, Rio Branco
Eastern Standard Time (Mexico)	(UTC-05:00) Chetumal
Eastern Standard Time	(UTC-05:00) Eastern Time (US & Canada)
Haiti Standard Time	(UTC-05:00) Haiti
Cuba Standard Time	(UTC-05:00) Havana
US Eastern Standard Time	(UTC-05:00) Indiana (East)
Turks And Caicos Standard Time	(UTC-05:00) Turks and Caicos
Paraguay Standard Time	(UTC-04:00) Asuncion
Atlantic Standard Time	(UTC-04:00) Atlantic Time (Canada)
Venezuela Standard Time	(UTC-04:00) Caracas
Central Brazilian Standard Time	(UTC-04:00) Cuiaba
SA Western Standard Time	(UTC-04:00) Georgetown, La Paz, Manaus, San Juan
Pacific SA Standard Time	(UTC-04:00) Santiago
Newfoundland Standard Time	(UTC-03:30) Newfoundland
Tocantins Standard Time	(UTC-03:00) Araguaina
E. South America Standard Time	(UTC-03:00) Brasilia
SA Eastern Standard Time	(UTC-03:00) Cayenne, Fortaleza

TIME ZONE ID	TIME ZONE DISPLAY NAME
Argentina Standard Time	(UTC-03:00) City of Buenos Aires
Greenland Standard Time	(UTC-03:00) Greenland
Montevideo Standard Time	(UTC-03:00) Montevideo
Magallanes Standard Time	(UTC-03:00) Punta Arenas
Saint Pierre Standard Time	(UTC-03:00) Saint Pierre and Miquelon
Bahia Standard Time	(UTC-03:00) Salvador
UTC-02	(UTC-02:00) Coordinated Universal Time-02
Mid-Atlantic Standard Time	(UTC-02:00) Mid-Atlantic - Old
Azores Standard Time	(UTC-01:00) Azores
Cape Verde Standard Time	(UTC-01:00) Cabo Verde Is.
UTC	(UTC) Coordinated Universal Time
GMT Standard Time	(UTC+00:00) Dublin, Edinburgh, Lisbon, London
Greenwich Standard Time	(UTC+00:00) Monrovia, Reykjavik
W. Europe Standard Time	(UTC+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna
Central Europe Standard Time	(UTC+01:00) Belgrade, Bratislava, Budapest, Ljubljana, Prague
Romance Standard Time	(UTC+01:00) Brussels, Copenhagen, Madrid, Paris
Morocco Standard Time	(UTC+01:00) Casablanca
Sao Tome Standard Time	(UTC+01:00) Sao Tome
Central European Standard Time	(UTC+01:00) Sarajevo, Skopje, Warsaw, Zagreb
W. Central Africa Standard Time	(UTC+01:00) West Central Africa
Jordan Standard Time	(UTC+02:00) Amman
GTB Standard Time	(UTC+02:00) Athens, Bucharest
Middle East Standard Time	(UTC+02:00) Beirut
Egypt Standard Time	(UTC+02:00) Cairo
E. Europe Standard Time	(UTC+02:00) Chisinau

TIME ZONE ID	TIME ZONE DISPLAY NAME
Syria Standard Time	(UTC+02:00) Damascus
West Bank Standard Time	(UTC+02:00) Gaza, Hebron
South Africa Standard Time	(UTC+02:00) Harare, Pretoria
FLE Standard Time	(UTC+02:00) Helsinki, Kyiv, Riga, Sofia, Tallinn, Vilnius
Israel Standard Time	(UTC+02:00) Jerusalem
Kaliningrad Standard Time	(UTC+02:00) Kaliningrad
Sudan Standard Time	(UTC+02:00) Khartoum
Libya Standard Time	(UTC+02:00) Tripoli
Namibia Standard Time	(UTC+02:00) Windhoek
Arabic Standard Time	(UTC+03:00) Baghdad
Turkey Standard Time	(UTC+03:00) Istanbul
Arab Standard Time	(UTC+03:00) Kuwait, Riyadh
Belarus Standard Time	(UTC+03:00) Minsk
Russian Standard Time	(UTC+03:00) Moscow, St. Petersburg
E. Africa Standard Time	(UTC+03:00) Nairobi
Iran Standard Time	(UTC+03:30) Tehran
Arabian Standard Time	(UTC+04:00) Abu Dhabi, Muscat
Astrakhan Standard Time	(UTC+04:00) Astrakhan, Ulyanovsk
Azerbaijan Standard Time	(UTC+04:00) Baku
Russia Time Zone 3	(UTC+04:00) Izhevsk, Samara
Mauritius Standard Time	(UTC+04:00) Port Louis
Saratov Standard Time	(UTC+04:00) Saratov
Georgian Standard Time	(UTC+04:00) Tbilisi
Volgograd Standard Time	(UTC+04:00) Volgograd
Caucasus Standard Time	(UTC+04:00) Yerevan

TIME ZONE ID	TIME ZONE DISPLAY NAME
Afghanistan Standard Time	(UTC+04:30) Kabul
West Asia Standard Time	(UTC+05:00) Ashgabat, Tashkent
Ekaterinburg Standard Time	(UTC+05:00) Ekaterinburg
Pakistan Standard Time	(UTC+05:00) Islamabad, Karachi
India Standard Time	(UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi
Sri Lanka Standard Time	(UTC+05:30) Sri Jayawardenepura
Nepal Standard Time	(UTC+05:45) Kathmandu
Central Asia Standard Time	(UTC+06:00) Astana
Bangladesh Standard Time	(UTC+06:00) Dhaka
Omsk Standard Time	(UTC+06:00) Omsk
Myanmar Standard Time	(UTC+06:30) Yangon (Rangoon)
SE Asia Standard Time	(UTC+07:00) Bangkok, Hanoi, Jakarta
Altai Standard Time	(UTC+07:00) Barnaul, Gorno-Altaysk
W. Mongolia Standard Time	(UTC+07:00) Hovd
North Asia Standard Time	(UTC+07:00) Krasnoyarsk
N. Central Asia Standard Time	(UTC+07:00) Novosibirsk
Tomsk Standard Time	(UTC+07:00) Tomsk
China Standard Time	(UTC+08:00) Beijing, Chongqing, Hong Kong, Urumqi
North Asia East Standard Time	(UTC+08:00) Irkutsk
Singapore Standard Time	(UTC+08:00) Kuala Lumpur, Singapore
W. Australia Standard Time	(UTC+08:00) Perth
Taipei Standard Time	(UTC+08:00) Taipei
Ulaanbaatar Standard Time	(UTC+08:00) Ulaanbaatar
Aus Central W. Standard Time	(UTC+08:45) Eucla
Transbaikal Standard Time	(UTC+09:00) Chita

TIME ZONE ID	TIME ZONE DISPLAY NAME
Tokyo Standard Time	(UTC+09:00) Osaka, Sapporo, Tokyo
North Korea Standard Time	(UTC+09:00) Pyongyang
Korea Standard Time	(UTC+09:00) Seoul
Yakutsk Standard Time	(UTC+09:00) Yakutsk
Cen. Australia Standard Time	(UTC+09:30) Adelaide
AUS Central Standard Time	(UTC+09:30) Darwin
E. Australia Standard Time	(UTC+10:00) Brisbane
AUS Eastern Standard Time	(UTC+10:00) Canberra, Melbourne, Sydney
West Pacific Standard Time	(UTC+10:00) Guam, Port Moresby
Tasmania Standard Time	(UTC+10:00) Hobart
Vladivostok Standard Time	(UTC+10:00) Vladivostok
Lord Howe Standard Time	(UTC+10:30) Lord Howe Island
Bougainville Standard Time	(UTC+11:00) Bougainville Island
Russia Time Zone 10	(UTC+11:00) Chokurdakh
Magadan Standard Time	(UTC+11:00) Magadan
Norfolk Standard Time	(UTC+11:00) Norfolk Island
Sakhalin Standard Time	(UTC+11:00) Sakhalin
Central Pacific Standard Time	(UTC+11:00) Solomon Is., New Caledonia
Russia Time Zone 11	(UTC+12:00) Anadyr, Petropavlovsk-Kamchatsky
New Zealand Standard Time	(UTC+12:00) Auckland, Wellington
UTC+12	(UTC+12:00) Coordinated Universal Time+12
Fiji Standard Time	(UTC+12:00) Fiji
Kamchatka Standard Time	(UTC+12:00) Petropavlovsk-Kamchatsky - Old
Chatham Islands Standard Time	(UTC+12:45) Chatham Islands
UTC+13	(UTC+13:00) Coordinated Universal Time+13

TIME ZONE ID	TIME ZONE DISPLAY NAME
Tonga Standard Time	(UTC+13:00) Nuku'alofa
Samoa Standard Time	(UTC+13:00) Samoa
Line Islands Standard Time	(UTC+14:00) Kiritimati Island

## See also

- [CURRENT\\_TIMEZONE \(Transact-SQL\)](#)
- [AT TIME ZONE \(Transact-SQL\)](#)
- [sys.time\\_zone\\_info \(Transact-SQL\)](#)

# Azure SQL Database managed instance connection types

11/7/2019 • 2 minutes to read • [Edit Online](#)

This article explains how clients connect to Azure SQL Database managed instance depending on the connection type. Script samples to change connection types are provided below, along with considerations related to changing the default connectivity settings.

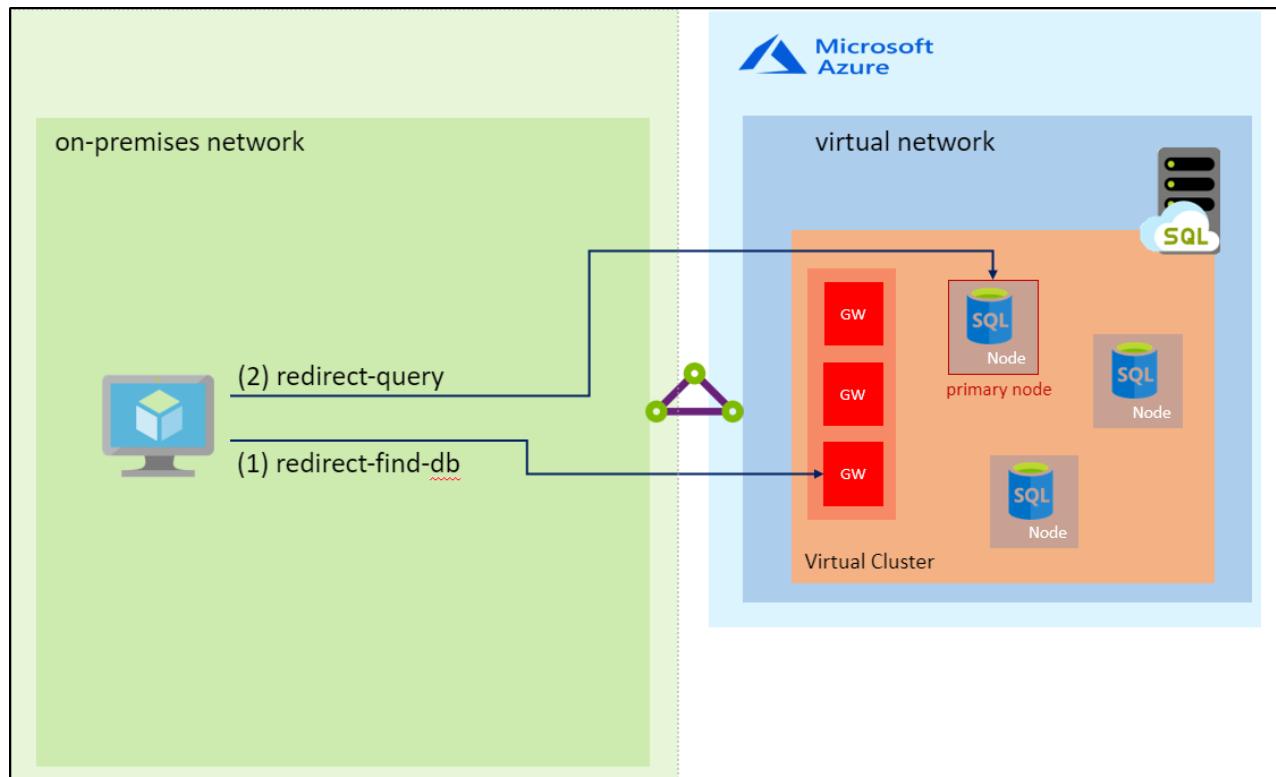
## Connection types

Azure SQL Database managed instance supports the following two connection types:

- **Redirect (recommended):** Clients establish connections directly to the node hosting the database. To enable connectivity using redirect, you must open firewalls and Network Security Groups (NSG) to allow access on ports 1433, and 11000-11999. Packets go directly to the database, and hence there are latency and throughput performance improvements using Redirect over Proxy.
- **Proxy (default):** In this mode, all connections are using a proxy gateway component. To enable connectivity, only port 1433 for private networks and port 3342 for public connection needs to be opened. Choosing this mode can result in higher latency and lower throughput, depending on nature of the workload. We highly recommend the Redirect connection policy over the Proxy connection policy for the lowest latency and highest throughput.

## Redirect connection type

Redirect connection type means that after the TCP session is established to the SQL engine, the client session obtains the destination virtual IP of the virtual cluster node from the load balancer. Subsequent packets flow directly to the virtual cluster node, bypassing the gateway. The following diagram illustrates this traffic flow.

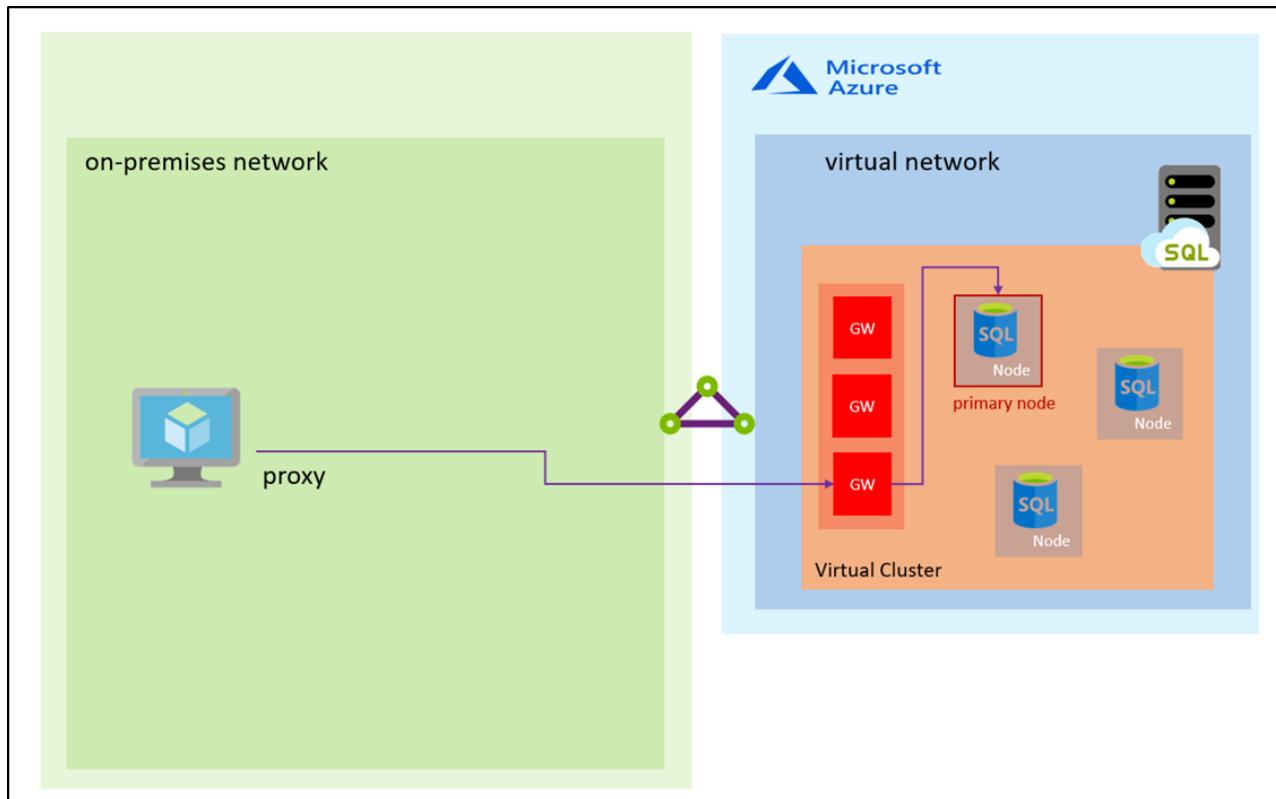


## IMPORTANT

Redirection connection type currently works only for private endpoint. Regardless of the connection type setting, connections coming through the public endpoint would be through a proxy.

## Proxy connection type

Proxy connection type means that the TCP session is established using the gateway and all subsequent packets flow through it. The following diagram illustrates this traffic flow.



## Script to change connection type settings using PowerShell

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

The following PowerShell script shows how to change the connection type for a managed instance to Redirect.

```
Install-Module -Name Az
Import-Module Az.Accounts
Import-Module Az.Sql

Connect-AzAccount
Get your SubscriptionId from the Get-AzSubscription command
Get-AzSubscription
Use your SubscriptionId in place of {subscription-id} below
Select-AzSubscription -SubscriptionId {subscription-id}
Replace {rg-name} with the resource group for your managed instance, and replace {mi-name} with the name of
your managed instance
$mi = Get-AzSqlInstance -ResourceGroupName {rg-name} -Name {mi-name}
$mi = $mi | Set-AzSqlInstance -ProxyOverride "Redirect" -force
```

## Next steps

- [Restore a database to a managed instance](#)
- Learn how to [configure a public endpoint on managed instance](#)
- Learn about [managed instance connectivity architecture](#)

# Migrate certificate of TDE protected database to Azure SQL Database Managed Instance

11/27/2019 • 4 minutes to read • [Edit Online](#)

When migrating a database protected by [Transparent Data Encryption](#) to Azure SQL Database Managed Instance using native restore option, the corresponding certificate from the on-premises or IaaS SQL Server needs to be migrated before database restore. This article walks you through the process of manual migration of the certificate to Azure SQL Database Managed Instance:

- Export certificate to a Personal Information Exchange (.pfx) file
- Extract certificate from file to base-64 string
- Upload it using PowerShell cmdlet

For an alternative option using fully managed service for seamless migration of both TDE protected database and corresponding certificate, see [How to migrate your on-premises database to Managed Instance using Azure Database Migration Service](#).

## IMPORTANT

Migrated certificate is used for restore of the TDE protected database only. Soon after restore is done, the migrated certificate gets replaced by a different protector, either service-managed certificate or asymmetric key from the key vault, depending on the type of the transparent data encryption you set on the instance.

## Prerequisites

To complete the steps in this article, you need the following prerequisites:

- [Pvk2Pfx](#) command-line tool installed on the on-premises server or other computer with access to the certificate exported as a file. Pvk2Pfx tool is part of the [Enterprise Windows Driver Kit](#), a standalone self-contained command-line environment.
- [Windows PowerShell](#) version 5.0 or higher installed.
- [PowerShell](#)
- [Azure CLI](#)

Make sure you have the following:

- Azure PowerShell module [installed and updated](#).
- [Az.Sql module](#).

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## IMPORTANT

The PowerShell Azure Resource Manager module is still supported by Azure SQL Database, but all future development is for the Az.Sql module. For these cmdlets, see [AzureRM.Sql](#). The arguments for the commands in the Az module and in the AzureRm modules are substantially identical.

Run the following commands in PowerShell to install/update the module:

```
Install-Module -Name Az.Sql
Update-Module -Name Az.Sql
```

## Export TDE certificate to a Personal Information Exchange (.pfx) file

The certificate can be exported directly from the source SQL Server, or from the certificate store if being kept there.

### Export certificate from the source SQL Server

Use the following steps to export certificate with SQL Server Management Studio and convert it into pfx format. Generic names *TDE\_Cert* and *full\_path* are being used for certificate and file names and paths through the steps. They should be replaced with the actual names.

1. In SSMS, open a new query window and connect to the source SQL Server.
2. Use the following script to list TDE protected databases and get the name of the certificate protecting encryption of the database to be migrated:

```
USE master
GO
SELECT db.name as [database_name], cer.name as [certificate_name]
FROM sys.dm_database_encryption_keys dek
LEFT JOIN sys.certificates cer
ON dek.encryptor_thumbprint = cer.thumbprint
INNER JOIN sys.databases db
ON dek.database_id = db.database_id
WHERE dek.encryption_state = 3
```

The screenshot shows the SSMS interface with a query window titled '1 - List Certificates...aster (demoUser (53))'. The query listed is the same as above. The results pane shows a table with three rows:

	database_name	certificate_name
1	tempdb	NULL
2	TDE_DB	TDE_Cert
3	TDE_DB2	TDE_Cert2

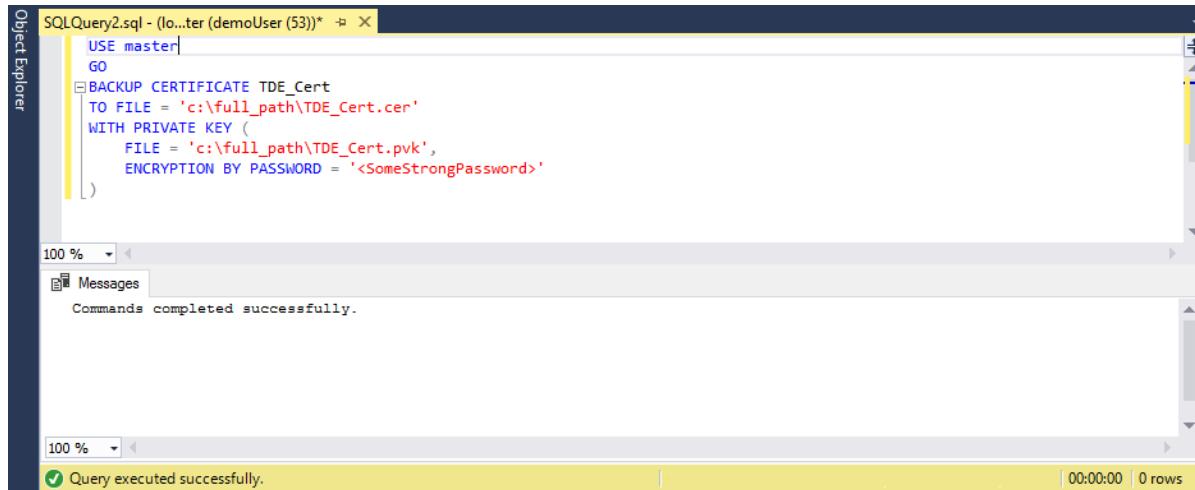
At the bottom of the results pane, a message says 'Query executed successfully.' and indicates 'master | 00:00:00 | 3 rows'.

3. Execute the following script to export the certificate to a pair of files (.cer and .pvk), keeping the public and private key information:

```

USE master
GO
BACKUP CERTIFICATE TDE_Cert
TO FILE = 'c:\full_path\TDE_Cert.cer'
WITH PRIVATE KEY (
 FILE = 'c:\full_path\TDE_Cert.pvk',
 ENCRYPTION BY PASSWORD = '<SomeStrongPassword>'
)

```



4. Use PowerShell console to copy certificate information from a pair of newly created files to a Personal Information Exchange (.pfx) file, using Pvk2Pfx tool:

```

.\pvk2pfx -pvk c:/full_path/TDE_Cert.pvk -pi "<SomeStrongPassword>" -spc c:/full_path/TDE_Cert.cer -
pfx c:/full_path/TDE_Cert.pfx

```

#### Export certificate from certificate store

If certificate is kept in SQL Server's local machine certificate store, it can be exported using the following steps:

1. Open PowerShell console and execute the following command to open Certificates snap-in of Microsoft Management Console:

```
certlm
```

2. In the Certificates MMC snap-in expand the path Personal -> Certificates to see the list of certificates
3. Right click certificate and click Export...
4. Follow the wizard to export certificate and private key to a Personal Information Exchange format

## Upload certificate to Azure SQL Database Managed Instance using Azure PowerShell cmdlet

- [PowerShell](#)
- [Azure CLI](#)

1. Start with preparation steps in PowerShell:

```
import the module into the PowerShell session
Import-Module Az
connect to Azure with an interactive dialog for sign-in
Connect-AzAccount
list subscriptions available and copy id of the subscription target Managed Instance belongs to
Get-AzSubscription
set subscription for the session
Select-AzSubscription <subscriptionId>
```

- Once all preparation steps are done, run the following commands to upload base-64 encoded certificate to the target Managed Instance:

```
$fileContentBytes = Get-Content 'C:/full_path/TDE_Cert.pfx' -Encoding Byte
$base64EncodedCert = [System.Convert]::ToString($fileContentBytes)
$securePrivateBlob = $base64EncodedCert | ConvertTo-SecureString -AsPlainText -Force
$password = "<password>"
$securePassword = $password | ConvertTo-SecureString -AsPlainText -Force
Add-AzSqlManagedInstanceTransparentDataEncryptionCertificate -ResourceGroupName "<resourceGroupName>" `
 -ManagedInstanceName "<managedInstanceName>" -PrivateBlob $securePrivateBlob -Password
$securePassword
```

The certificate is now available to the specified Managed Instance and backup of corresponding TDE protected database can be restored successfully.

## Next steps

In this article, you learned how to migrate certificate protecting encryption key of database with Transparent Data Encryption, from the on-premises or IaaS SQL Server to Azure SQL Database Managed Instance.

See [Restore a database backup to an Azure SQL Database Managed Instance](#) to learn how to restore a database backup to an Azure SQL Database Managed Instance.

# Restore a SQL database in a managed instance to a previous point in time

1/22/2020 • 5 minutes to read • [Edit Online](#)

Use point-in-time restore (PITR) to create a database as a copy of another database from some time in the past. This article describes how to do a point-in-time restore of a database in an Azure SQL Database managed instance.

Point-in-time restore is useful in recovery scenarios, such as incidents caused by errors, incorrectly loaded data, or deletion of crucial data. You can also use it simply for testing or auditing. Backup files are kept for 7 to 35 days, depending on your database settings.

Point-in-time restore can:

- Restore a database from an existing database.
- Restore a database from a deleted database.

For a managed instance, point-in-time restore can also:

- Restore a database to the same managed instance.
- Restore a database to another managed instance.

## NOTE

Point-in-time restore of a whole managed instance is not possible. This article explains only what's possible: point-in-time restore of a database that's hosted on a managed instance.

## Limitations

When you're restoring from one managed instance to another, both instances must be in the same subscription and region. Cross-region and cross-subscription restore aren't currently supported.

## WARNING

Be aware of the storage size of your managed instance. Depending on size of the data to be restored, you might run out of instance storage. If there isn't enough space for the restored data, use a different approach.

The following table shows point-in-time restore scenarios for managed instances:

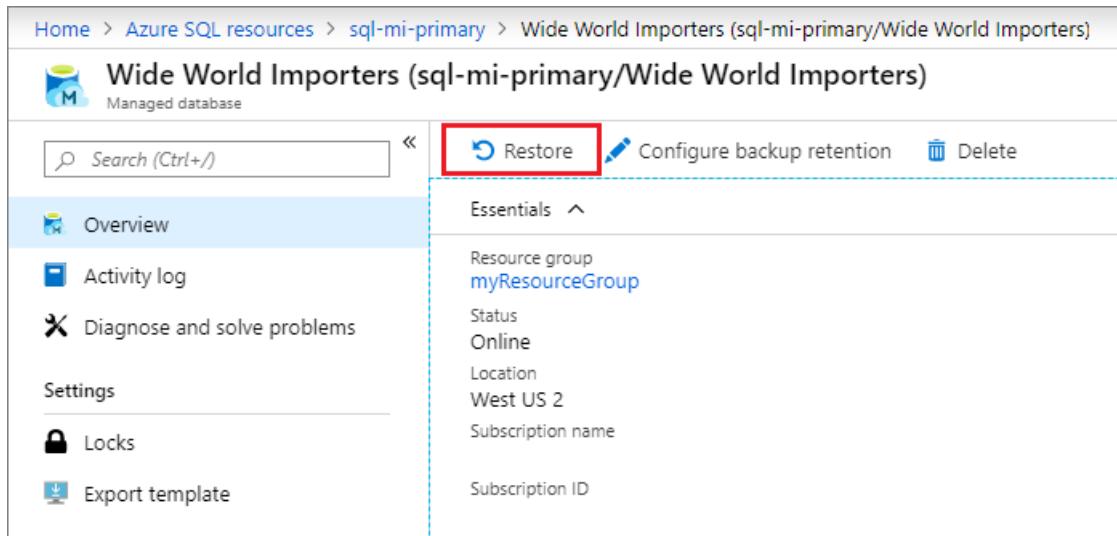
	RESTORE EXISTING DB TO SAME MANAGED INSTANCE	RESTORE EXISTING DB TO ANOTHER MANAGED INSTANCE	RESTORE DROPPED DB TO SAME MANAGED INSTANCE	RESTORE DROPPED DB TO ANOTHER MANAGED INSTANCE
Azure portal	Yes	No	No	No
Azure CLI	Yes	Yes	No	No
PowerShell	Yes	Yes	Yes	Yes

## Restore an existing database

Restore an existing database to the same instance by using the Azure portal, Powershell, or the Azure CLI. To restore a database to another instance, use Powershell or the Azure CLI so you can specify the properties for the target managed instance and resource group. If you don't specify these parameters, the database will be restored to the existing instance by default. The Azure portal doesn't currently support restoring to another instance.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the [Azure portal](#).
2. Go to your managed instance and select the database that you want to restore.
3. Select **Restore** on the database page:



4. On the **Restore** page, select the point for the date and time that you want to restore the database to.
5. Select **Confirm** to restore your database. This action starts the restore process, which creates a new database and populates it with data from the original database at the specified point in time. For more information about the recovery process, see [Recovery time](#).

## Restore a deleted database

Restoring a deleted database can be done by using PowerShell or Azure Portal. Please use this document to do this by [Azure Portal](#). The database can be restored to the same instance or another instance.

To restore a deleted database by using PowerShell, specify your values for the parameters in the following command. Then, run the command:

```

$subscriptionId = "<Subscription ID>"

Get-AzSubscription -SubscriptionId $subscriptionId

Select-AzSubscription -SubscriptionId $subscriptionId

$resourceGroupName = "<Resource group name>"

$managedInstanceName = "<Managed instance name>"

$deletedDatabaseName = "<Source database name>"

$deleted_db = Get-AzSqlDeletedInstanceDatabaseBackup -ResourceGroupName $resourceGroupName `

 -InstanceName $managedInstanceName -DatabaseName $deletedDatabaseName

$pointInTime = "2018-06-27T08:51:39.3882806Z"

$properties = New-Object System.Object

$properties | Add-Member -type NoteProperty -name CreateMode -Value "PointInTimeRestore"

$properties | Add-Member -type NoteProperty -name RestorePointInTime -Value $pointInTime

$properties | Add-Member -type NoteProperty -name RestorableDroppedDatabaseId -Value $deleted_db.Id

```

To restore the deleted database to another instance, change the names of the resource group and managed instance. Also, make sure that the location parameter matches the location of the resource group and the managed instance.

```

$resourceGroupName = "<Second resource group name>"

$managedInstanceName = "<Second managed instance name>"

$location = "West Europe"

$restoredDBName = "WorldWideImportersPITR"

$resource_id =

"subscriptions/$subscriptionId/resourceGroups/$resourceGroupName/providers/Microsoft.Sql/managedInstances/$managedInstanceName/databases/$restoredDBName"

New-AzResource -Location $location -Properties $properties `

 -ResourceId $resource_id -ApiVersion "2017-03-01-preview" -Force

```

## Overwrite an existing database

To overwrite an existing database, you must:

1. Drop the existing database that you want to overwrite.
2. Rename the point-in-time-restored database to the name of the database that you dropped.

### Drop the original database

You can drop the database by using the Azure portal, PowerShell, or the Azure CLI.

You can also drop the database by connecting to the managed instance directly, starting SQL Server Management Studio (SSMS), and then running the following Transact-SQL (T-SQL) command:

```
DROP DATABASE WorldWideImporters;
```

Use one of the following methods to connect to your database in the managed instance:

- [SSMS/Azure Data Studio via an Azure virtual machine](#)
- [Point-to-site](#)
- [Public endpoint](#)
- [Portal](#)
- [PowerShell](#)

- Azure CLI

In the Azure portal, select the database from the managed instance, and then select **Delete**.

The screenshot shows the Azure portal interface for managing a database named 'Wide World Importers' under a managed instance 'sql-mi-primary'. The top navigation bar includes 'Home', 'Azure SQL resources', 'sql-mi-primary', and the specific database name. Below the navigation is a search bar and a toolbar with 'Restore', 'Configure backup retention', and 'Delete' (which is highlighted with a red box). The main content area is titled 'Essentials' and displays the following information:

- Resource group:** myResourceGroup
- Status:** Online
- Location:** West US 2
- Subscription name:** (Subscription ID is also listed below it)

On the left side, there's a sidebar with links: Overview, Activity log, Diagnose and solve problems, Settings (which is currently selected), Locks, and Export template.

#### Alter the new database name to match the original database name

Connect directly to the managed instance and start SQL Server Management Studio. Then, run the following Transact-SQL (T-SQL) query. The query will change the name of the restored database to that of the dropped database that you intend to overwrite.

```
ALTER DATABASE WorldWideImportersPITR MODIFY NAME = WorldWideImporters;
```

Use one of the following methods to connect to your database in the managed instance:

- [Azure virtual machine](#)
- [Point-to-site](#)
- [Public endpoint](#)

## Next steps

Learn about [automated backups](#).

# Determine VNet subnet size for Azure SQL Database Managed Instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

Azure SQL Database Managed Instance must be deployed within an Azure [virtual network \(VNet\)](#).

The number of Managed Instances that can be deployed in the subnet of VNet depends on the size of the subnet (subnet range).

When you create a Managed Instance, Azure allocates a number of virtual machines depending on the tier you selected during provisioning. Because these virtual machines are associated with your subnet, they require IP addresses. To ensure high availability during regular operations and service maintenance, Azure may allocate additional virtual machines. As a result, the number of required IP addresses in a subnet is larger than the number of Managed Instances in that subnet.

By design, a Managed Instance needs a minimum of 16 IP addresses in a subnet and may use up to 256 IP addresses. As a result, you can use a subnet masks between /28 and /24 when defining your subnet IP ranges. A network mask bit of /28 (14 hosts per network) is a good size for a single general purpose or business-critical deployment. A mask bit of /27 (30 hosts per network) is ideal for a multiple Managed Instance deployments within the same VNet. Mask bit settings of /26 (62 hosts) and /24 (254 hosts) allows further scaling out of the VNet to support additional Managed Instances.

## IMPORTANT

A subnet size with 16 IP addresses is the bare minimum with limited potential where a scaling operation like vCore size change is not supported. Choosing subnet with the prefix /27 or longest prefix is highly recommended.

## Determine subnet size

If you plan to deploy multiple Managed Instances inside the subnet and need to optimize on subnet size, use these parameters to form a calculation:

- Azure uses five IP addresses in the subnet for its own needs
- Each General Purpose instance needs two addresses
- Each Business Critical instance needs four addresses

**Example:** You plan to have three General Purpose and two Business Critical Managed Instances. That means you need  $5 + 3 * 2 + 2 * 4 = 19$  IP addresses. As IP ranges are defined in power of 2, you need the IP range of 32 ( $2^5$ ) IP addresses. Therefore, you need to reserve the subnet with subnet mask of /27.

## IMPORTANT

Calculation displayed above will become obsolete with further improvements.

## Next steps

- For an overview, see [What is a Managed Instance](#).
- Learn more about [Connectivity architecture for Managed Instance](#).
- See how to [create VNet where you will deploy Managed Instances](#)

- For DNS issues, see [Configuring a Custom DNS](#)

# Create a virtual network for Azure SQL Database Managed Instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

This article explains how to create a valid virtual network and subnet where you can deploy Azure SQL Database Managed Instance.

Azure SQL Database Managed Instance must be deployed within an Azure [virtual network](#). This deployment enables the following scenarios:

- Secure private IP address
- Connecting to a Managed Instance directly from an on-premises network
- Connecting a Managed Instance to linked server or another on-premises data store
- Connecting a Managed Instance to Azure resources

## NOTE

You should [determine the size of the subnet for Managed Instance](#) before you deploy the first instance. You can't resize the subnet after you put the resources inside.

If you plan to use an existing virtual network, you need to modify that network configuration to accommodate your Managed Instance. For more information, see [Modify an existing virtual network for Managed Instance](#).

After a managed instance is created, moving the managed instance or VNet to another resource group or subscription is not supported.

## Create a virtual network

The easiest way to create and configure a virtual network is to use an Azure Resource Manager deployment template.

1. Sign in to the Azure portal.
2. Select the **Deploy to Azure** button:



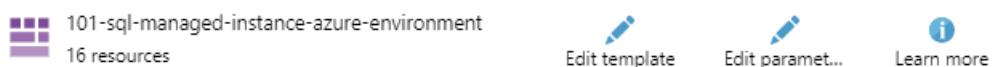
This button opens a form that you can use to configure the network environment where you can deploy Managed Instance.

## NOTE

This Azure Resource Manager template will deploy a virtual network with two subnets. One subnet, called **ManagedInstances**, is reserved for Managed Instance and has a preconfigured route table. The other subnet, called **Default**, is used for other resources that should access Managed Instance (for example, Azure Virtual Machines).

3. Configure the network environment. On the following form, you can configure parameters of your network environment:

## TEMPLATE



### BASICS

* Subscription	<input type="text"/>
* Resource group	<input type="text"/> Select a resource group <a href="#">Create new</a>
* Location	<input type="text"/> (Europe) West Europe

### SETTINGS

Virtual Network Name ⓘ	<input type="text"/> MyNewVNet
Virtual Network Address Prefix ⓘ	<input type="text"/> 10.0.0.0/16
Default Subnet Name ⓘ	<input type="text"/> Default
Default Subnet Prefix ⓘ	<input type="text"/> 10.0.0.0/24
Managed Instance Subnet Name ⓘ	<input type="text"/> ManagedInstances
Managed Instance Subnet Prefix ⓘ	<input type="text"/> 10.0.1.0/24
Route Table To Azure Service ⓘ	<input type="text"/> RouteToAzureSqlMiMngSvc
Nsg For Managed Instance Subnet ⓘ	<input type="text"/> nsg-ManagedInstances
Location ⓘ	<input type="text"/> [resourceGroup().location]

### TERMS AND CONDITIONS

[Template information](#) | [Azure Marketplace Terms](#) | [Azure Marketplace](#)

By clicking "Purchase," I (a) agree to the applicable legal terms associated with the offering; (b) authorize Microsoft to charge or bill my current payment method for the fees associated with the offering(s), including applicable taxes, with the same billing frequency as my Azure subscription, until I discontinue use of the offering(s); and (c) agree that, if the deployment involves 3rd party offerings, Microsoft may share my contact information and other details of such deployment with the publisher of that offering.

I agree to the terms and conditions stated above

**Purchase**

You might change the names of the virtual network and subnets, and adjust the IP ranges associated with your networking resources. After you select the **Purchase** button, this form will create and configure your environment. If you don't need two subnets, you can delete the default one.

## Next steps

- For an overview, see [What is a Managed Instance?](#).
- Learn about [connectivity architecture in Managed Instance](#).
- Learn how to [modify an existing virtual network for Managed Instance](#).
- For a tutorial that shows how to create a virtual network, create a Managed Instance, and restore a database from a database backup, see [Create an Azure SQL Database Managed Instance](#).
- For DNS issues, see [Configuring a custom DNS](#).

# Configure an existing virtual network for Azure SQL Database Managed Instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

Azure SQL Database Managed Instance must be deployed within an Azure [virtual network](#) and the subnet dedicated for Managed Instances only. You can use the existing virtual network and subnet if it's configured according to the [Managed Instance virtual network requirements](#).

If one of the following cases applies to you, you can validate and modify your network by using the script explained in this article:

- You have a new subnet that's still not configured.
- You're not sure that the subnet is aligned with the [requirements](#).
- You want to check that the subnet still complies with the [network requirements](#) after you made changes.

## NOTE

You can create a Managed Instance only in virtual networks created through the Azure Resource Manager deployment model. Azure virtual networks created through the classic deployment model are not supported. Calculate subnet size by following the guidelines in the [Determine the size of subnet for Managed Instances](#) article. You can't resize the subnet after you deploy the resources inside.

After a managed instance is created, moving the managed instance or VNet to another resource group or subscription is not supported.

## Validate and modify an existing virtual network

If you want to create a Managed Instance inside an existing subnet, we recommend the following PowerShell script to prepare the subnet:

```
$scriptUrlBase = 'https://raw.githubusercontent.com/Microsoft/sql-server-samples/master/samples/manage/azure-sql-db-managed-instance/prepare-subnet'

$parameters = @{
 subscriptionId = '<subscriptionId>'
 resourceGroupName = '<resourceGroupName>'
 virtualNetworkName = '<virtualNetworkName>'
 subnetName = '<subnetName>'
}

Invoke-Command -ScriptBlock ([Scriptblock]::Create((iwr ($scriptUrlBase+'/prepareSubnet.ps1?t='+[DateTime]::Now.Ticks)).Content)) -ArgumentList $parameters
```

The script prepares the subnet in three steps:

1. Validate: It validates the selected virtual network and subnet for Managed Instance networking requirements.
2. Confirm: It shows the user a set of changes that need to be made to prepare the subnet for Managed Instance deployment. It also asks for consent.
3. Prepare: It properly configures the virtual network and subnet.

## Next steps

- For an overview, see [What is a Managed Instance?](#).
- For a tutorial that shows how to create a virtual network, create a Managed Instance, and restore a database from a database backup, see [Create an Azure SQL Database Managed Instance](#).
- For DNS issues, see [Configuring a custom DNS](#).

# Delete a subnet after deleting an Azure SQL Database managed instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

This article provides guidelines on how to manually delete a subnet after deleting the last Azure SQL Database managed instance residing in it.

Managed instances are deployed into [virtual clusters](#). Each virtual cluster is associated with a subnet. The virtual cluster persists by design for 12 hours after the last instance deletion to enable you to more quickly create managed instances in the same subnet. There's no charge for keeping an empty virtual cluster. During this period, the subnet associated with the virtual cluster can't be deleted.

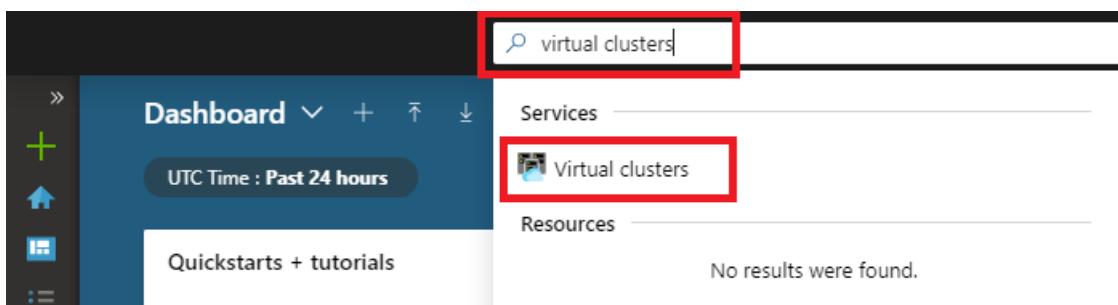
If you don't want to wait 12 hours and prefer to delete the virtual cluster and its subnet sooner, you can do so manually. Delete the virtual cluster manually by using the Azure portal or the virtual clusters API.

## IMPORTANT

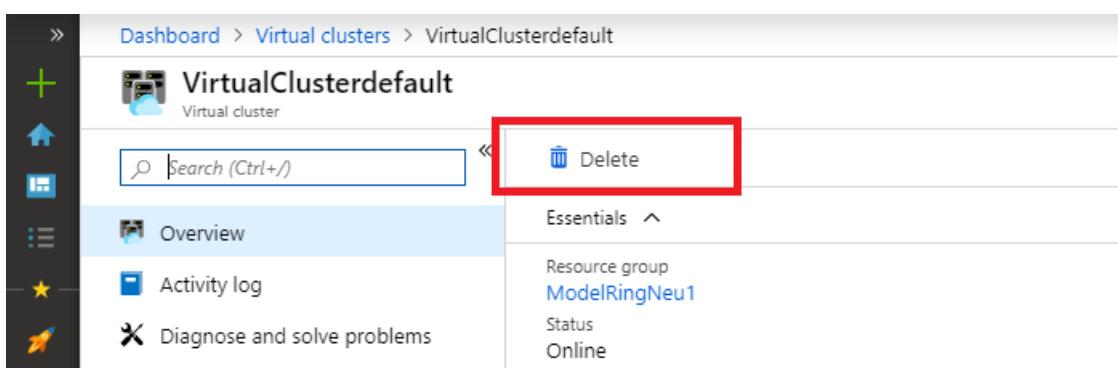
- The virtual cluster should contain no managed instances for the deletion to be successful.
- Deletion of a virtual cluster is a long running operation lasting for about 1.5 hours (see [Managed instance management operations](#) for up to date virtual cluster delete time) during which the virtual cluster will still be visible in the portal until this process is completed.

## Delete virtual cluster from the Azure portal

To delete a virtual cluster by using the Azure portal, search for the virtual cluster resources.



After you locate the virtual cluster you want to delete, select this resource, and select **Delete**. You're prompted to confirm the virtual cluster deletion.



Azure portal notifications will show you a confirmation that request to delete the virtual cluster has been successfully submitted. Deletion operation itself will last for about 1.5 hours during which the virtual cluster will

still be visible in portal. Once the process is completed, the virtual cluster will no longer be visible and the subnet associated with it will be released for reuse.

**TIP**

If there are no managed instances shown in the virtual cluster, and you are unable to delete the virtual cluster, ensure that you do not have an ongoing instance deployment in progress. This includes started and canceled deployments that are still in progress. This is because these operations will still use the virtual cluster locking it from deletion. Reviewing Deployments tab of the resource group the instance was deployed to will indicate any deployments in progress. In this case, wait for the deployment to complete, delete managed instance and then the virtual cluster.

## Delete virtual cluster by using the API

To delete a virtual cluster through the API, use the URI parameters specified in the [virtual clusters delete method](#).

## Next steps

- For an overview, see [What is a Managed Instance?](#).
- Learn about [connectivity architecture in Managed Instance](#).
- Learn how to [modify an existing virtual network for Managed Instance](#).
- For a tutorial that shows how to create a virtual network, create a Managed Instance, and restore a database from a database backup, see [Create an Azure SQL Database Managed Instance](#).
- For DNS issues, see [Configuring a custom DNS](#).

# Configuring a Custom DNS for Azure SQL Database Managed Instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

An Azure SQL Database Managed Instance must be deployed within an Azure [virtual network \(VNet\)](#). There are a few scenarios (for example, db mail, linked servers to other SQL instances in your cloud or hybrid environment) that require private host names to be resolved from the Managed Instance. In this case, you need to configure a custom DNS inside Azure.

Because Managed Instance uses the same DNS for its inner workings, configure the custom DNS server so that it can resolve public domain names.

## IMPORTANT

Always use a fully qualified domain name (FQDN) for the mail server, the SQL Server instance, and for other services, even if they're within your private DNS zone. For example, use `smtp.contoso.com` for your mail server because `smtp` won't resolve correctly. Creating a linked server or replication that references SQL VMs inside the same virtual network also requires an FQDN and a default DNS suffix. For example, `SQLVM.internal.cloudapp.net`. For more information, see [Name resolution that uses your own DNS server](#).

## IMPORTANT

Updating virtual network DNS servers won't affect Managed Instance immediately. Managed Instance DNS configuration is updated after the DHCP lease expires or after the platform upgrade, whichever occurs first. **Users are advised to set their virtual network DNS configuration before creating their first Managed Instance.**

## Next steps

- For an overview, see [What is a Managed Instance](#)
- For a tutorial showing you how to create a new Managed Instance, see [Creating a Managed Instance](#).
- For information about configuring a VNet for a Managed Instance, see [VNet configuration for Managed Instances](#)

# Sync networking configuration for Azure App Service hosting plan

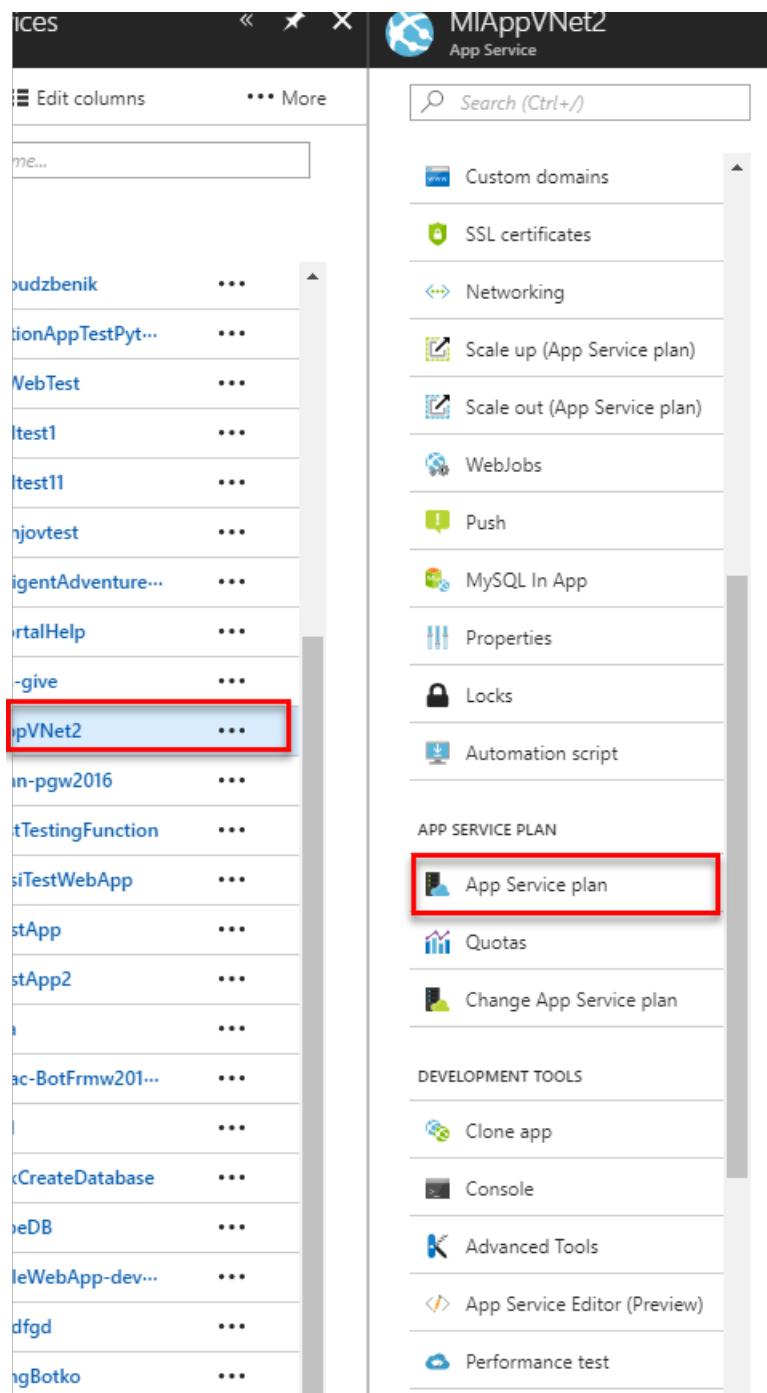
11/6/2019 • 2 minutes to read • [Edit Online](#)

It might happen that although you [integrated your app with an Azure Virtual Network](#), you can't establish connection to Managed Instance. One thing you can try is to refresh networking configuration for your service plan.

## Sync network configuration for App Service hosting plan

To do that, follow these steps:

1. Go to your web apps App Service plan.



2. Click **Networking** and then click **Click here to Manage**.

The screenshot shows the 'MITestApp - Networking' blade in the Azure portal. On the left, a sidebar lists various management options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (with Apps, File system storage, and Networking selected), Scale up (App Service plan), Scale out (App Service plan), Properties, Locks, and Automation script. A red box highlights the 'Networking' option. On the right, there are two main sections: 'VNET Integration' (loading) and 'Hybrid connections'. The 'VNET Integration' section includes a 'Click here to manage' button, which is also highlighted with a red box.

3. Select your **VNet** and click **Sync Network**.

The screenshot shows the 'Virtual Network Integration' blade for the 'MITestApp' app service plan. It displays the following information:

App Service Plan	MITestApp
App Service Plan Location	East US
VNETs integrated with	2 out of 5
VNET NAME	GATEWAY STATUS
VNET_MIPlayground_Peer	Online
vNet1-eastus	Online

On the right side, there are tabs for Save, Discard, and Sync Network (which is highlighted with a red box). The 'Sync Network' tab shows the following details:

VNET NAME	VNET_MIPlayground_Peer
LOCATION	East US
CERTIFICATE STATUS	Certificates in sync
GATEWAY STATUS	Online

Below these are sections for APPS USING VIRTUAL NETWORK (MITestApp2) and VNET ADDRESS SPACE (START ADDRESS: 10.3.0.0, END ADDRESS: 10.3.0.255).

4. Wait until the sync is done.

## Notifications

X

Dismiss: [Informational](#) [Completed](#) [All](#)

 Synchronized Virtual Network Certificates	12:21
Certificates have been synced	
 Successfully synchronized routes	12:21
All routes were synchronized between Apps and Virtual Network	
 Synchronized Virtual Network Data	12:20
Virtual Network data has been synced	

You are now ready to try to re-establish your connection to your Managed Instance.

## Next steps

- For information about configuring your VNet for Managed Instance, see [Managed Instance VNet architecture](#) and [How to configure existing VNet](#).

# Determine the management endpoint IP address

11/7/2019 • 2 minutes to read • [Edit Online](#)

The Azure SQL Database Managed Instance virtual cluster contains a management endpoint that Microsoft uses for management operations. The management endpoint is protected with a built-in firewall on the network level and mutual certificate verification on the application level. You can determine the IP address of the management endpoint, but you can't access this endpoint.

To determine the management IP address, do a DNS lookup on your managed instance FQDN:

`mi-name.zone_id.database.windows.net` . This will return a DNS entry that's like  
`trx.region-a.worker.vnet.database.windows.net` . You can then do a DNS lookup on this FQDN with ".vnet" removed. This will return the management IP address.

This PowerShell will do it all for you if you replace <MI FQDN> with the DNS entry of your managed instance:

`mi-name.zone_id.database.windows.net` :

```
$MIFQDN = "<MI FQDN>"
resolve-dnsname $MIFQDN | select -first 1 | %{ resolve-dnsname $_.NameHost.Replace(".vnet","")}
```

For more information about Managed Instances and connectivity, see [Azure SQL Database Managed Instance Connectivity Architecture](#).

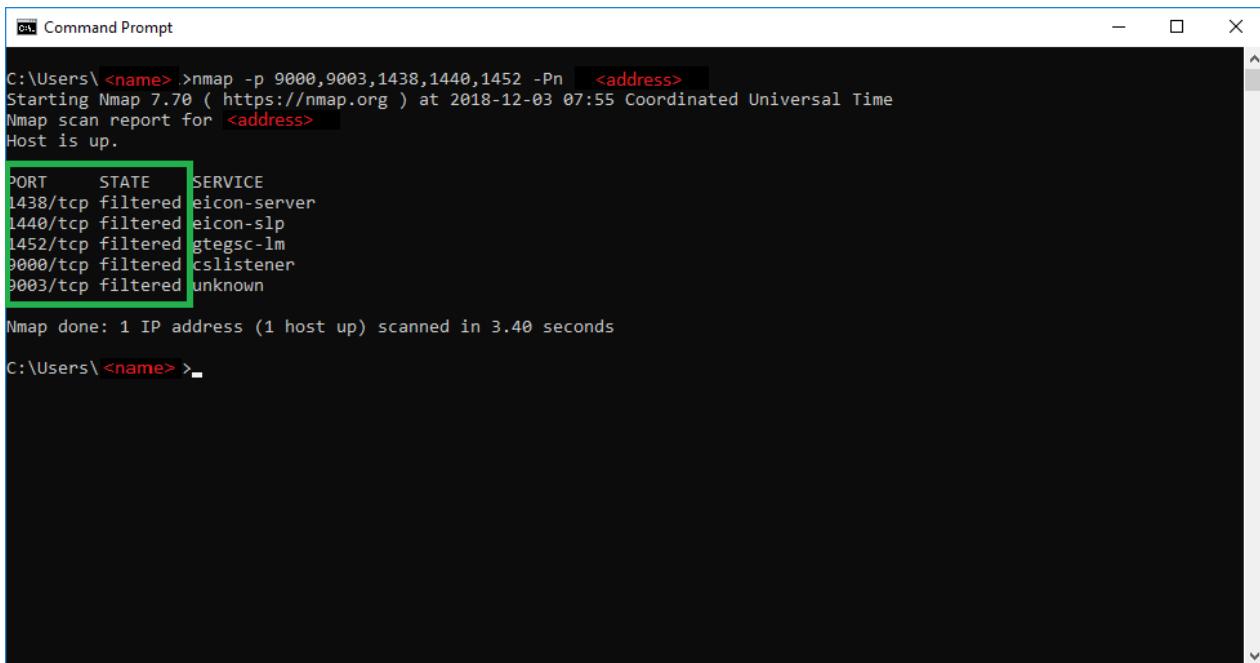
# Verifying the Managed Instance built-in firewall

11/7/2019 • 2 minutes to read • [Edit Online](#)

The Managed Instance [mandatory inbound security rules](#) require management ports 9000, 9003, 1438, 1440, 1452 to be open from **Any source** on the Network Security Group (NSG) that protects the Managed Instance. Although these ports are open at the NSG level, they are protected at the network level by the built-in firewall.

## Verify firewall

To verify these ports, use any security scanner tool to test these ports. The following screenshot shows how to use one of these tools.



A screenshot of a Windows Command Prompt window titled "Command Prompt". The command entered is "nmap -p 9000,9003,1438,1440,1452 -Pn <address>". The output shows the following results:

```
C:\Users\<name>>nmap -p 9000,9003,1438,1440,1452 -Pn <address>
Starting Nmap 7.70 (https://nmap.org) at 2018-12-03 07:55 Coordinated Universal Time
Nmap scan report for <address>
Host is up.

PORT STATE SERVICE
1438/tcp filtered eicon-server
1440/tcp filtered eicon-slp
1452/tcp filtered gtegsc-lm
9000/tcp filtered cslistener
9003/tcp filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 3.40 seconds
C:\Users\<name>>-
```

## Next steps

For more information about Managed Instances and connectivity, see [Azure SQL Database Managed Instance Connectivity Architecture](#).

# Configure Advanced Threat Protection in Azure SQL Database managed instance

11/7/2019 • 2 minutes to read • [Edit Online](#)

Advanced Threat Protection for a [managed instance](#) detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases. Advanced Threat Protection can identify **Potential SQL injection**, **Access from unusual location or data center**, **Access from unfamiliar principal or potentially harmful application**, and **Brute force SQL credentials** - see more details in [Advanced Threat Protection alerts](#).

You can receive notifications about the detected threats via [email notifications](#) or [Azure portal](#)

Advanced Threat Protection is part of the [advanced data security](#) (ADS) offering, which is a unified package for advanced SQL security capabilities. Advanced Threat Protection can be accessed and managed via the central SQL ADS portal.

## Set up Advanced Threat Protection in the Azure portal

1. Launch the Azure portal at <https://portal.azure.com>.
2. Navigate to the configuration page of the managed instance you want to protect. In the **Settings** page, select **Advanced Data Security**.
3. In the Advanced Data Security configuration page
  - Turn **ON** Advanced Data Security.
  - Configure the **list of emails** to receive security alerts upon detection of anomalous database activities.
  - Select the **Azure storage account** where anomalous threat audit records are saved.
  - Select the **Advanced Threat Protection types** that you would like configured. Learn more about [Advanced Threat Protection alerts](#).
4. Click **Save** to save the new or updated Advanced Data Security policy.

sql-mi-primary - Advanced Data Security

SQL managed instance

Search (Ctrl+ /)

Save Discard Feedback

OVERVIEW

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Settings

Quick start

Connection strings

Active Directory admin

Pricing tier

Instance Failover Groups (pr...)

Properties

Locks

Export template

Security

Advanced Data Security

Virtual network

Transparent data encryption ...

Monitoring

Metrics

ADVANCED DATA SECURITY

ON OFF

Advanced Data Security costs 15 USD/managed instance/month. It includes Data Discovery & Classification, Vulnerability Assessment and Advanced Threat Protection. We invite you to a trial period for the first 30 days, without charge.

VULNERABILITY ASSESSMENT SETTINGS

Subscription SQL DB Content >

Storage account >

Periodic recurring scans ON OFF

Send scan reports to

Also send email notification to admins and subscription owners

ADVANCED THREAT PROTECTION SETTINGS

Send alerts to  Email addresses

Also send email notification to admins and subscription owners

Advanced Threat Protection types > All

#### NOTE

Prices in screenshots does not always reflect the current price, and are an example.

## Next steps

- Learn more about [Advanced Threat Protection](#).
- Learn about managed instances, see [What is a managed instance](#).
- Learn more about [Advanced Threat Protection for single database](#).
- Learn more about [managed instance auditing](#).
- Learn more about [Azure security center](#).

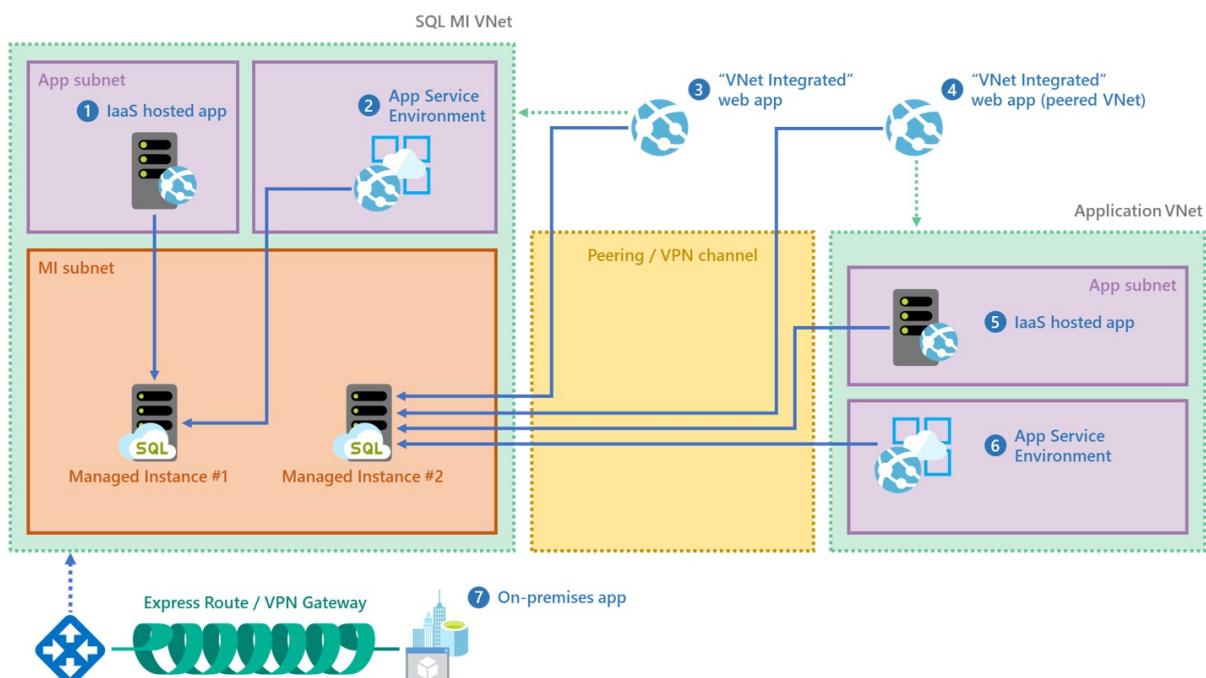
# Connect your application to Azure SQL Database managed instance

11/7/2019 • 5 minutes to read • [Edit Online](#)

Today you have multiple choices when deciding how and where you host your application.

You may choose to host application in the cloud either by using Azure App Service or some of Azure's virtual network (VNet) integrated options like Azure App Service Environment, Virtual Machine, Virtual Machine Scale Set. You could also take hybrid cloud approach and keep your applications on-premises.

Whatever choice you made, you can connect it to a Managed Instance.



## Connect an application inside the same VNet

This scenario is the simplest. Virtual machines inside the VNet can connect to each other directly even if they are inside different subnets. That means that all you need to connect application inside an Azure Application Environment or Virtual Machine is to set the connection string appropriately.

## Connect an application inside a different VNet

This scenario is a bit more complex because Managed Instance has private IP address in its own VNet. To connect, an application needs access to the VNet where Managed Instance is deployed. So, first you need to make a connection between the application and the Managed Instance VNet. The VNets don't have to be in the same subscription in order for this scenario to work.

There are two options for connecting VNets:

- [Azure Virtual Network peering](#)
- [VNet-to-VNet VPN gateway \(Azure portal, PowerShell, Azure CLI\)](#)

The peering option is the preferable one because peering uses the Microsoft backbone network so, from the connectivity perspective, there is no noticeable difference in latency between virtual machines in peered VNet

and in the same VNet. VNet peering is limited to the networks in the same region.

#### IMPORTANT

VNet peering scenario for Managed Instance is limited to the networks in the same region due to [constraints of the Global Virtual Network peering](#). See also the relevant section of the [Azure Virtual Networks Frequently Asked Questions](#) article for more details.

## Connect an on-premises application

Managed Instance can only be accessed through a private IP address. In order to access it from on-premises, you need to make a Site-to-Site connection between the application and the Managed Instance VNet.

There are two options how to connect on-premises to Azure VNet:

- Site-to-Site VPN connection ([Azure portal](#), [PowerShell](#), [Azure CLI](#))
- ExpressRoute connection

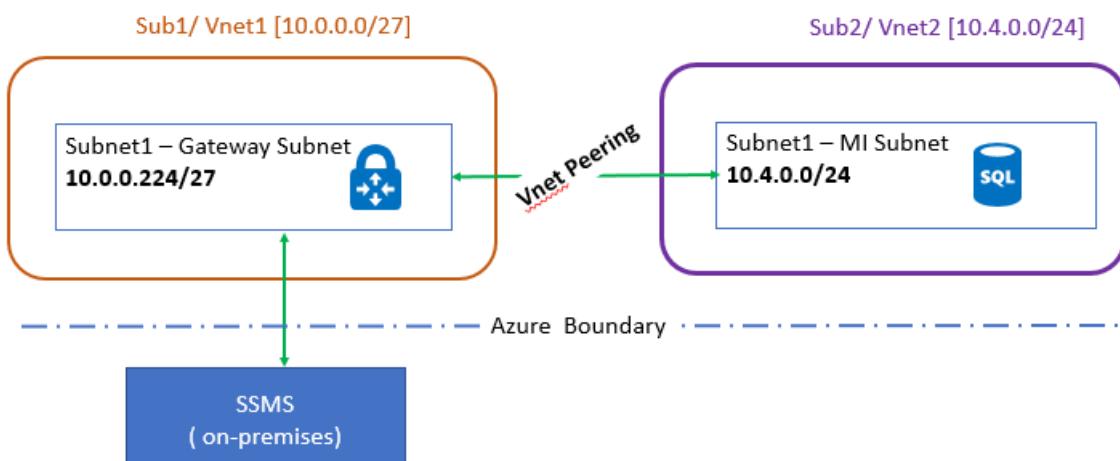
If you've established on-premises to Azure connection successfully and you can't establish connection to Managed Instance, check if your firewall has open outbound connection on SQL port 1433 as well as 11000-11999 range of ports for redirection.

## Connect an application on the developers box

Managed Instance can be accessed only through a private IP address so in order to access it from your developer box, you first need to make a connection between your developer box and the Managed Instance VNet. To do so, configure a Point-to-Site connection to a VNet using native Azure certificate authentication. For more information, see [Configure a point-to-site connection to connect to an Azure SQL Database Managed Instance from on-premises computer](#).

## Connect from on-premises with VNet peering

Another scenario implemented by customers is where VPN gateway is installed in a separate virtual network and a subscription from the one hosting Managed Instance. The two virtual networks are then peered. The following sample architecture diagram shows how this can be implemented.



Once you have the basic infrastructure set up, you need to modify some setting so that the VPN Gateway can see the IP addresses in the virtual network that hosts the Managed Instance. To do so, make the following very specific changes under the **Peering settings**.

1. In the VNet that hosts the VPN gateway, go to **Peerings**, then to the Managed Instance peered VNet connection, and then click **Allow Gateway Transit**.
2. In the VNet that hosts the Managed Instance, go to **Peerings**, then to the VPN Gateway peered VNet connection, and then click **Use remote gateways**.

## Connect an Azure App Service hosted application

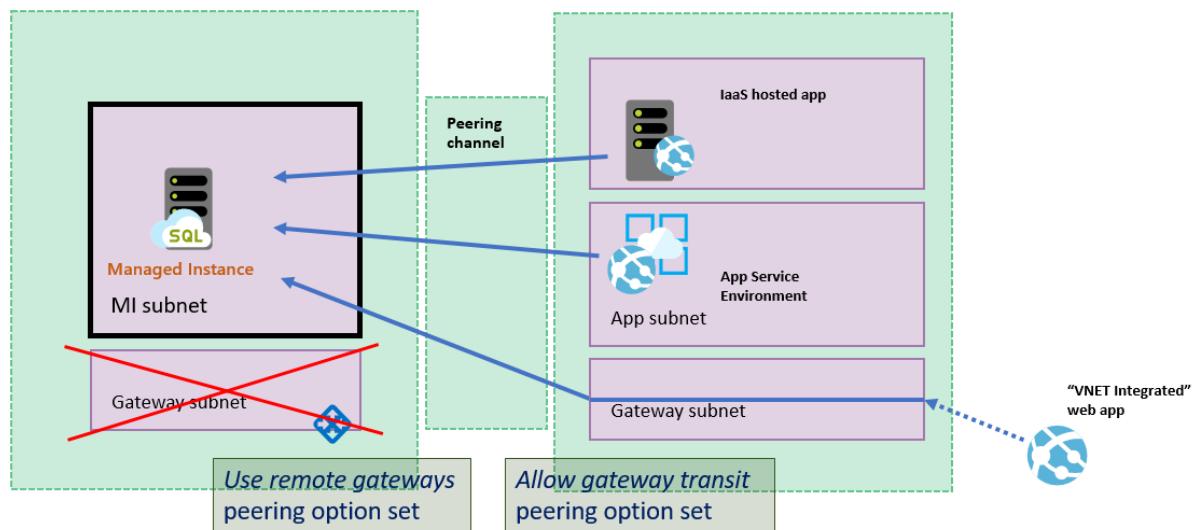
Managed Instance can be accessed only through a private IP address so in order to access it from Azure App Service you first need to make a connection between the application and the Managed Instance VNet. See [Integrate your app with an Azure Virtual Network](#).

For troubleshooting, see [Troubleshooting VNets and Applications](#). If a connection cannot be established, try [synching the networking configuration](#).

A special case of connecting Azure App Service to Managed Instance is when you integrated Azure App Service to a network peered to Managed Instance VNet. That case requires the following configuration to be set up:

- Managed Instance VNet must NOT have gateway
- Managed Instance VNet must have Use remote gateways option set
- Peered VNet must have Allow gateway transit option set

This scenario is illustrated in the following diagram:



### NOTE

The VNet Integration feature does not integrate an app with a VNet that has an ExpressRoute Gateway. Even if the ExpressRoute Gateway is configured in coexistence mode the VNet Integration does not work. If you need to access resources through an ExpressRoute connection, then you can use an App Service Environment, which runs in your VNet.

## Troubleshooting connectivity issues

For troubleshooting connectivity issues, review the following:

- If you are unable to connect to Managed Instance from an Azure virtual machine within the same VNet but different subnet, check if you have a Network Security Group set on VM subnet that might be blocking access. Additionally note that you need to open outbound connection on SQL port 1433 as well as ports in range 11000-11999 since those are needed for connecting via redirection inside the Azure

boundary.

- Ensure that BGP Propagation is set to **Enabled** for the route table associated with the VNet.
- If using P2S VPN, check the configuration in the Azure portal to see if you see **Ingress/Egress** numbers. Non-zero numbers indicate that Azure is routing traffic to/from on-premises.

## Connection health

Connections	1
Ingress (bytes)	375208
Egress (bytes)	594811

## Address pool

172.26.34.0/24

## Tunnel type

SSL VPN (SSTP)

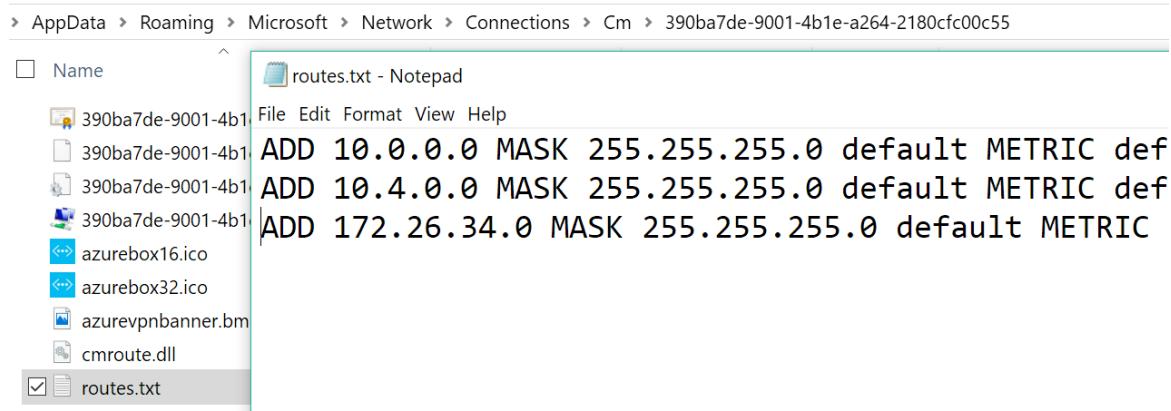
IKEv2 VPN

## Authentication type

Azure certificate  RADIUS authentication

- Check that the client machine (that is running the VPN client) has route entries for all the VNets that you need to access. The routes are stored in

`%AppData%\ Roaming\Microsoft\Network\Connections\Cm\<GUID>\routes.txt`.



As shown in this image, there are two entries for each VNet involved and a third entry for the VPN endpoint that is configured in the Portal.

Another way to check the routes is via the following command. The output shows the routes to the various subnets:

```
C:\>route print -4
=====
Interface List
14...54 ee 75 67 6b 39Intel(R) Ethernet Connection (3) I218-LM
57.....rndatavnet
18...94 65 9c 7d e5 ceIntel(R) Dual Band Wireless-AC 7265
1.....Software Loopback Interface 1
Adapter=-----

IPv4 Route Table
=====
Active Routes:
Network Destination Netmask Gateway Interface Metric
 0.0.0.0 0.0.0.0 10.83.72.1 10.83.74.112 35
 10.0.0.0 255.255.255.0 On-link 172.26.34.2 43
 10.4.0.0 255.255.255.0 On-link 172.26.34.2 43
=====
Persistent Routes:
None
```

- If using VNet peering, ensure that you have followed the instructions for setting [Allow Gateway Transit and Use Remote Gateways](#).

## Required versions of drivers and tools

The following minimal versions of the tools and drivers are recommended if you want to connect to Managed Instance:

Driver/Tool	Version
.NET Framework	4.6.1 (or .NET Core)
ODBC driver	v17
PHP driver	5.2.0
JDBC driver	6.4.0
Node.js driver	2.1.1
OLEDB driver	18.0.2.0
SSMS	18.0 or <a href="#">higher</a>
<a href="#">SMO</a>	<a href="#">150</a> or higher

## Next steps

- For information about Managed Instance, see [What is a Managed Instance](#).
- For a tutorial showing you how to create a new Managed Instance, see [Create a Managed Instance](#).

# Azure CLI samples for Azure SQL Database

11/7/2019 • 2 minutes to read • [Edit Online](#)

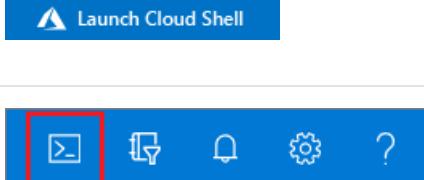
Azure SQL Database can be configured using [Azure CLI](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the CLI locally, this topic requires that you are running the Azure CLI version 2.0 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).

- [Single database & Elastic pools](#)
- [Managed Instance](#)

The following table includes links to Azure CLI script examples for Azure SQL Database.

<a href="#">Create a single database and an elastic pool</a>	
<a href="#">Create a single database and configure a firewall rule</a>	This CLI script example creates a single Azure SQL database and configures a server-level firewall rule.

Create elastic pools and move pooled databases	This CLI script example creates SQL elastic pools, and moves pooled Azure SQL databases, and changes compute sizes.
<b>Scale a single database and an elastic pool</b>	
Scale a single database	This CLI script example scales a single Azure SQL database to a different compute size after querying the size information for the database.
Scale an elastic pool	This CLI script example scales a SQL elastic pool to a different compute size.
<b>Failover groups</b>	
Add single database to failover group	This CLI script creates a database, and a failover group, adds the database to the failover group and tests failover to the secondary server.

Learn more about the [Single Database Azure CLI API](#).

# Azure PowerShell samples for Azure SQL Database

11/7/2019 • 4 minutes to read • [Edit Online](#)

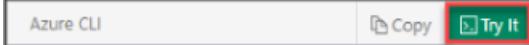
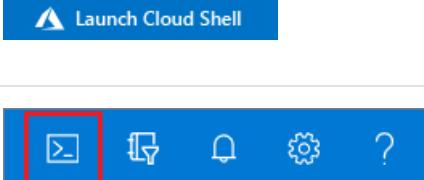
Azure SQL Database enables you to configure your databases, instances, and pools using Azure PowerShell.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the PowerShell locally, this tutorial requires AZ PowerShell 1.4.0 or later. If you need to upgrade, see [Install Azure PowerShell module](#). If you are running PowerShell locally, you also need to run `Connect-AzAccount` to create a connection with Azure.

- [Single Database and Elastic pools](#)
- [Managed Instance](#)

The following table includes links to sample Azure PowerShell scripts for Azure SQL Database.

<b>Create and configure single databases, and elastic pools</b>	
-----------------------------------------------------------------	--

<a href="#">Create a single database and configure a database server firewall rule</a>	This PowerShell script creates a single Azure SQL database and configures a server-level firewall rule.
<a href="#">Create elastic pools and move pooled databases</a>	This PowerShell script creates Azure SQL Database elastic pools, and moves pooled databases, and changes compute sizes.
<b>Configure geo-replication and failover</b>	
<a href="#">Configure and failover a single database using active geo-replication</a>	This PowerShell script configures active geo-replication for a single Azure SQL database and fails it over to the secondary replica.
<a href="#">Configure and failover a pooled database using active geo-replication</a>	This PowerShell script configures active geo-replication for an Azure SQL database in a SQL elastic pool, and fails it over to the secondary replica.
<b>Configure a failover group</b>	
<a href="#">Configure a failover group for a single database</a>	This PowerShell script creates a database, and a failover group, adds the database to the failover group and tests failover to the secondary server.
<a href="#">Configure a failover group for an elastic pool</a>	This PowerShell script creates a database, adds it to an elastic pool, adds the elastic pool to the failover group and tests failover to the secondary server.
<b>Scale a single database and an elastic pool</b>	
<a href="#">Scale a single database</a>	This PowerShell script monitors the performance metrics of an Azure SQL database, scales it to a higher compute size and creates an alert rule on one of the performance metrics.
<a href="#">Scale an elastic pool</a>	This PowerShell script monitors the performance metrics of an Azure SQL Database elastic pool, scales it to a higher compute size, and creates an alert rule on one of the performance metrics.
<b>Auditing and threat detection</b>	
<a href="#">Configure auditing and threat-detection</a>	This PowerShell script configures auditing and threat detection policies for an Azure SQL database.
<b>Restore, copy, and import a database</b>	
<a href="#">Restore a database</a>	This PowerShell script restores an Azure SQL database from a geo-redundant backup and restores a deleted Azure SQL database to the latest backup.
<a href="#">Copy a database to new server</a>	This PowerShell script creates a copy of an existing Azure SQL database in a new Azure SQL server.
<a href="#">Import a database from a bacpac file</a>	This PowerShell script imports a database to an Azure SQL server from a bacpac file.

<b>Sync data between databases</b>	
<a href="#">Sync data between SQL databases</a>	This PowerShell script configures Data Sync to sync between multiple Azure SQL databases.
<a href="#">Sync data between SQL Database and SQL Server on-premises</a>	This PowerShell script configures Data Sync to sync between an Azure SQL database and a SQL Server on-premises database.
<a href="#">Update the SQL Data Sync sync schema</a>	This PowerShell script adds or removes items from the Data Sync sync schema.

Learn more about the [Single Database Azure PowerShell API](#).

## Additional resources

The examples listed on this page use the [Azure SQL Database cmdlets](#) for creating and managing Azure SQL resources. Additional cmdlets for running queries, and performing many database tasks are located in the [sqlserver](#) module. For more information, see [SQL Server PowerShell](#).

# Azure Resource Manager templates for Azure SQL Database

11/7/2019 • 3 minutes to read • [Edit Online](#)

Azure Resource Manager templates enable you to define your infrastructure as code and deploy your solutions to Azure cloud.

- [Single database & elastic pool](#)
- [Managed Instance](#)

The following table includes links to Azure Resource Manager templates for Azure SQL Database.

<a href="#">Single database</a>	This Azure Resource Manager template creates a single Azure SQL Database with logical server and configures firewall rules.
<a href="#">Logical server</a>	This Azure Resource Manager template creates a logical server for Azure SQL Database.
<a href="#">Elastic pool</a>	This template allows you to deploy a new Elastic pool with its new associated SQL Server and new SQL Databases to assign to it.
<a href="#">Failover groups</a>	This template creates two Azure SQL logical servers, a SQL database, and a failover group.
<a href="#">Threat Detection</a>	This template allows you to deploy an Azure SQL logical server and a set of Azure SQL Databases with Threat Detection enabled, with an email address for alerts for each database. Threat Detection is part of the SQL Advanced Threat Protection (ATP) offering and provides a layer of security that responds to potential threats over SQL servers and databases.
<a href="#">Auditing to Azure Blob Storage</a>	This template allows you to deploy an Azure SQL logical server with Auditing enabled to write audit logs to a blob storage. Auditing for Azure SQL Database tracks database events and writes them to an audit log that can be placed in your Azure storage account, OMS workspace, or Event Hubs.
<a href="#">Auditing to Azure Event Hub</a>	This template allows you to deploy an Azure SQL server with Auditing enabled to write audit logs to an existing Event Hub. In order to send audit events to Event Hub, set auditing settings with <code>Enabled</code> <code>State</code> and set <code>IsAzureMonitorTargetEnabled</code> as <code>true</code> . Also, configure Diagnostic Settings with <code>SQLSecurityAuditEvents</code> diagnostic logs category on the <code>master</code> database (for server level auditing). Auditing for Azure SQL Database and SQL Data Warehouse tracks database events and writes them to an audit log that can be placed in your Azure storage account, OMS workspace, or Event Hubs.

Azure Web App with SQL Database	This sample creates a free Azure Web App and SQL Database at the "Basic" service level.
Azure Web App and Redis Cache with SQL Database	This template creates a Web App, Redis Cache, and SQL Database in the same resource group, and creates two connection strings in the Web App for the SQL Database and Redis Cache.
Import data from blob storage using ADF V2	This Azure Resource Manager template creates Azure Data Factory V2 that copies data from Azure Blob Storage to SQL Database.
HDInsight cluster with a SQL Database	This template allows you to create a HDInsight cluster, a SQL Database server, a SQL Database, and two tables. This template is used by the <a href="#">Use Sqoop with Hadoop in HDInsight article</a>
Azure Logic App that runs a SQL Stored Procedure on a schedule	This template allows you to create a Logic App that will run a SQL stored procedure on schedule. Any arguments for the procedure can be put into the body section of the template.

# Azure SQL Database glossary of terms

1/23/2020 • 2 minutes to read • [Edit Online](#)

CONTEXT	TERM	MORE INFORMATION
Azure service	Azure SQL Database or SQL Database	<a href="#">The Azure SQL Database service</a>
Purchasing model	DTU-based purchasing model	<a href="#">DTU-based purchasing model</a>
	vCore-based purchasing model	<a href="#">vCore-based purchasing model</a>
Deployment option	Single database	<a href="#">Single databases</a>
	Elastic pool	<a href="#">Elastic pool</a>
	Managed instance	<a href="#">Managed instance</a>
Service tier	Basic, Standard, Premium, General Purpose, Hyperscale, Business Critical	For service tiers in the vCore model, see <a href="#">single database and elastic pool</a> and <a href="#">managed instance</a> . For service tiers in the DTU model, see <a href="#">DTU model</a> .
Compute tier	Serverless compute	<a href="#">Serverless compute</a>
	Provisioned compute	<a href="#">Provisioned compute</a>
Compute generation	Gen5, M-series, Fsv2-series	<a href="#">Hardware generations</a>
Server entity	SQL Database server or database server	<a href="#">Database server</a>
	SQL Database managed instance server, managed instance server, or instance server	<a href="#">Managed instance</a>
Resource type	vCore	A CPU core provided to the compute resource for a single database, elastic pool, or managed instance.
	Compute size and storage amount	Compute size is the maximum amount of CPU, memory and other non-storage related resources available for a single database, elastic pool, or managed instance. Storage size is the maximum amount of storage available for a single database, elastic pool, or managed instance. For sizing options in the vcore model, see <a href="#">vCore single databases</a> , <a href="#">vCore elastic pools</a> and <a href="#">managed instances</a> . For sizing options in the DTU model, see <a href="#">DTU single databases</a> and <a href="#">DTU elastic pools</a> .

# Public data sets for testing and prototyping

12/10/2019 • 4 minutes to read • [Edit Online](#)

Browse this list of public data sets for data that you can use to prototype and test storage and analytics services and solutions.

## U.S. Government and agency data

DATA SOURCE	ABOUT THE DATA	ABOUT THE FILES
<a href="#">US Government data</a>	Over 250,000 data sets covering agriculture, climate, consumer, ecosystems, education, energy, finance, health, local government, manufacturing, maritime, ocean, public safety, and science and research in the U.S.	Files of various sizes in various formats including HTML, XML, CSV, JSON, Excel, and many others. You can filter available data sets by file format.
<a href="#">US Census data</a>	Statistical data about the population of the U.S.	Data sets are in various formats.
<a href="#">Earth science data from NASA</a>	Over 32,000 data collections covering agriculture, atmosphere, biosphere, climate, cryosphere, human dimensions, hydrosphere, land surface, oceans, sun-earth interactions, and more.	Data sets are in various formats.
<a href="#">Airline flight delays and other transportation data</a>	"The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics (BTS) tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights appears ... in summary tables posted on this website."	Files are in CSV format.
<a href="#">Traffic fatalities - US Fatality Analysis Reporting System (FARS)</a>	"FARS is a nationwide census providing NHTSA, Congress, and the American public yearly data regarding fatal injuries suffered in motor vehicle traffic crashes."	"Create your own fatality data run online by using the FARS Query System. Or download all FARS data from 1975 to present from the FTP Site."
<a href="#">Toxic chemical data - EPA Toxicity ForeCaster (ToxCast™) data</a>	"EPA's most updated, publicly available high-throughput toxicity data on thousands of chemicals. This data is generated through the EPA's ToxCast research effort."	Data sets are available in various formats including spreadsheets, R packages, and MySQL database files.

DATA SOURCE	ABOUT THE DATA	ABOUT THE FILES
Toxic chemical data - NIH Tox21 Data Challenge 2014	"The 2014 Tox21 data challenge is designed to help scientists understand the potential of the chemicals and compounds being tested through the Toxicology in the 21st Century initiative to disrupt biological pathways in ways that may result in toxic effects."	Data sets are available in SMILES and SDF formats. The data provides "assay activity data and chemical structures on the Tox21 collection of ~10,000 compounds (Tox21 10K)."
Biotechnology and genome data from the NCBI	Multiple data sets covering genes, genomes, and proteins.	Data sets are in text, XML, BLAST, and other formats. A BLAST app is available.

## Other statistical and scientific data

DATA SOURCE	ABOUT THE DATA	ABOUT THE FILES
New York City taxi data	"Taxi trip records include fields capturing pick-up and dropoff dates/times, pick-up and dropoff locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts."	Data sets are in CSV files by month.
Microsoft Research data sets - "Data Science for Research"	Multiple data sets covering human-computer interaction, audio/video, data mining/information retrieval, geospatial/location, natural language processing, and robotics/computer vision.	Data sets are in various formats, zipped for download.
Public genome data	"A diverse data set of whole human genomes are freely available for public use to enhance any genomic study..." The provider, Complete Genomics, is a private for-profit corporation.	Data sets, after extraction, are in UNIX text format. Analysis tools are also available.
Open Science Data Cloud data	"The Open Science Data Cloud provides the scientific community with resources for storing, sharing, and analyzing terabyte and petabyte-scale scientific datasets."	Data sets are in various formats.
Global climate data - WorldClim	"WorldClim is a set of global climate layers (gridded climate data) with a spatial resolution of about 1 km2. These data can be used for mapping and spatial modeling."	These files contain geospatial data. For more info, see <a href="#">Data format</a> .
Data about human society - The GDELT Project	"The GDELT Project is the largest, most comprehensive, and highest resolution open database of human society ever created."	The raw data files are in CSV format.
Advertising click prediction data for machine learning from Criteo	"The largest ever publicly released ML dataset." For more info, see <a href="#">Criteo's 1 TB Click Prediction Dataset</a> .	

DATA SOURCE	ABOUT THE DATA	ABOUT THE FILES
ClueWeb09 text mining data set from The Lemur Project	"The ClueWeb09 dataset was created to support research on information retrieval and related human language technologies. It consists of about 1 billion web pages in 10 languages that were collected in January and February 2009."	See <a href="#">Dataset Information</a> .

## Online service data

DATA SOURCE	ABOUT THE DATA	ABOUT THE FILES
GitHub archive	"GitHub Archive is a project to record the public GitHub timeline [of events], archive it, and make it easily accessible for further analysis."	Download JSON-encoded event archives in .gz (Gzip) format from a web client.
GitHub activity data from The GHTorrent project	"The GHTorrent project [is] an effort to create a scalable, queryable, offline mirror of data offered through the GitHub REST API. GHTorrent monitors the GitHub public event time line. For each event, it retrieves its contents and their dependencies, exhaustively."	MySQL database dumps are in CSV format.
Stack Overflow data dump	"This is an anonymized dump of all user-contributed content on the Stack Exchange network [including Stack Overflow]."	"Each site [such as Stack Overflow] is formatted as a separate archive consisting of XML files zipped via 7-zip using bzip2 compression. Each site archive includes Posts, Users, Votes, Comments, PostHistory, and PostLinks."

# Troubleshooting connectivity issues and other errors with Microsoft Azure SQL Database

2/25/2020 • 24 minutes to read • [Edit Online](#)

You receive error messages when the connection to Azure SQL Database fails. These connection problems can be caused by Azure SQL Database reconfiguration, firewall settings, a connection timeout, incorrect login information or failure to apply best practices and design guidelines during the [application design] (sql-database-develop-overview.md) process. Additionally, if the maximum limit on some Azure SQL Database resources is reached, you can't connect to Azure SQL Database.

## Transient fault error messages (40197, 40613 and others)

The Azure infrastructure has the ability to dynamically reconfigure servers when heavy workloads arise in the SQL Database service. This dynamic behavior might cause your client program to lose its connection to SQL Database. This kind of error condition is called a *transient fault*. Database reconfiguration events occur because of a planned event (for example, a software upgrade) or an unplanned event (for example, a process crash, or load balancing). Most reconfiguration events are generally short-lived and should be completed in less than 60 seconds at most. However, these events can occasionally take longer to finish, such as when a large transaction causes a long-running recovery. The following table lists various transient errors that applications can receive when connecting to SQL Database

### List of transient fault error codes

ERROR CODE	SEVERITY	DESCRIPTION
4060	16	Cannot open database "%.*ls" requested by the login. The login failed. For more information, see <a href="#">Errors 4000 to 4999</a>

ERROR CODE	SEVERITY	DESCRIPTION
40197	17	<p>The service has encountered an error processing your request. Please try again. Error code %d.</p> <p>You receive this error when the service is down due to software or hardware upgrades, hardware failures, or any other failover problems. The error code (%d) embedded within the message of error 40197] provides additional information about the kind of failure or failover that occurred. Some examples of the error codes are embedded within the message of error 40197 are 40020, 40143, 40166, and 40540.</p> <p>Reconnecting to your SQL Database server automatically connects you to a healthy copy of your database. Your application must catch error 40197, log the embedded error code (%d) within the message for troubleshooting, and try reconnecting to SQL Database until the resources are available, and your connection is established again. For more information, see <a href="#">Transient errors</a>.</p>
40501	20	<p>The service is currently busy. Retry the request after 10 seconds. Incident ID: %ls. Code: %d. For more information, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">Database server resource limits</a></li> <li>• <a href="#">DTU-based limits for single databases</a></li> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for single databases</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a></li> <li>• <a href="#">Managed instance resource limits</a>.</li> </ul>
40613	17	<p>Database '%.*ls' on server '%.*ls' is not currently available. Please retry the connection later. If the problem persists, contact customer support, and provide them the session tracing ID of '%.*ls'.</p> <p>This error may occur if there is already an existing dedicated administrator connection (DAC) established to the database. For more information, see <a href="#">Transient errors</a>.</p>

ERROR CODE	SEVERITY	DESCRIPTION
49918	16	<p>Cannot process request. Not enough resources to process request.</p> <p>The service is currently busy. Please retry the request later. For more information, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">Database server resource limits</a></li> <li>• <a href="#">DTU-based limits for single databases</a></li> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for single databases</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a></li> <li>• <a href="#">Managed instance resource limits</a>.</li> </ul>
49919	16	<p>Cannot process create or update request. Too many create or update operations in progress for subscription "%ld".</p> <p>The service is busy processing multiple create or update requests for your subscription or server. Requests are currently blocked for resource optimization. Query <a href="#">sys.dm_operation_status</a> for pending operations. Wait until pending create or update requests are complete or delete one of your pending requests and retry your request later. For more information, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">Database server resource limits</a></li> <li>• <a href="#">DTU-based limits for single databases</a></li> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for single databases</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a></li> <li>• <a href="#">Managed instance resource limits</a>.</li> </ul>
49920	16	<p>Cannot process request. Too many operations in progress for subscription "%ld".</p> <p>The service is busy processing multiple requests for this subscription. Requests are currently blocked for resource optimization. Query <a href="#">sys.dm_operation_status</a> for operation status. Wait until pending requests are complete or delete one of your pending requests and retry your request later. For more information, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">Database server resource limits</a></li> <li>• <a href="#">DTU-based limits for single databases</a></li> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for single databases</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a></li> <li>• <a href="#">Managed instance resource limits</a>.</li> </ul>

ERROR CODE	SEVERITY	DESCRIPTION
4221	16	Login to read-secondary failed due to long wait on 'HADR_DATABASE_WAIT_FOR_TRANSITION_TO_VERSIONING'. The replica is not available for login because row versions are missing for transactions that were in-flight when the replica was recycled. The issue can be resolved by rolling back or committing the active transactions on the primary replica. Occurrences of this condition can be minimized by avoiding long write transactions on the primary.

### Steps to resolve transient connectivity issues

1. Check the [Microsoft Azure Service Dashboard](#) for any known outages that occurred during the time during which the errors were reported by the application.
2. Applications that connect to a cloud service such as Azure SQL Database should expect periodic reconfiguration events and implement retry logic to handle these errors instead of surfacing these as application errors to users.
3. As a database approaches its resource limits, it can seem to be a transient connectivity issue. See [Resource limits](#).
4. If connectivity problems continue, or if the duration for which your application encounters the error exceeds 60 seconds or if you see multiple occurrences of the error in a given day, file an Azure support request by selecting **Get Support** on the [Azure Support](#) site.

### Implementing Retry Logic

It is strongly recommended that your client program has retry logic so that it could reestablish a connection after giving the transient fault time to correct itself. We recommend that you delay for 5 seconds before your first retry. Retrying after a delay shorter than 5 seconds risks overwhelming the cloud service. For each subsequent retry the delay should grow exponentially, up to a maximum of 60 seconds.

For code examples of retry logic, see:

- [Connect resiliently to SQL with ADO.NET](#)
- [Connect resiliently to SQL with PHP](#)

For additional information on handling transient errors in your application review

- [Troubleshooting transient connection errors to SQL Database](#)

A discussion of the *blocking period* for clients that use ADO.NET is available in [SQL Server Connection Pooling \(ADO.NET\)](#).

## A network-related or instance-specific error occurred while establishing a connection to SQL Database server

The issue occurs if the application can't connect to the server.

To resolve this issue, try the steps (in the order presented) in the [Steps to fix common connection issues](#) section.

## The server-instance was not found or was not accessible (errors 26, 40, 10053)

#### Error 26: Error Locating server specified

System.Data.SqlClient.SqlException: A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections.(provider: SQL Network Interfaces, error: 26 – Error Locating Server/Instance Specified)

#### Error 40: Could not open a connection to the server

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

#### Error 10053: A transport-level error has occurred when receiving results from the server

10053: A transport-level error has occurred when receiving results from the server. (Provider: TCP Provider, error: 0 - An established connection was aborted by the software in your host machine)

These issues occur if the application can't connect to the server.

To resolve these issues, try the steps (in the order presented) in the [Steps to fix common connection issues](#) section.

## Cannot connect to server due to firewall issues

#### Error 40615: Cannot connect to <servername>

To resolve this issue, [configure firewall settings on SQL Database through the Azure portal](#).

#### Error 5: Cannot connect to <servername>

To resolve this issue, make sure that port 1433 is open for outbound connections on all firewalls between the client and the internet.

For more information, see [Configure the Windows Firewall to allow SQL Server access](#).

## Unable to log in to the server (errors 18456, 40531)

#### Login failed for user '< User name >'

Login failed for user '<User name>'.This session has been assigned a tracing ID of '<Tracing ID>'. Provide this tracing ID to customer support when you need assistance. (Microsoft SQL Server, Error: 18456)

To resolve this issue, contact your service administrator to provide you with a valid SQL Server user name and password.

Typically, the service administrator can use the following steps to add the login credentials:

1. Log in to the server by using SQL Server Management Studio (SSMS).
2. Run the following SQL query to check whether the login name is disabled:

```
SELECT name, is_disabled FROM sys.sql_logins
```

3. If the corresponding name is disabled, enable it by using the following statement:

```
Alter login <User name> enable
```

4. If the SQL login user name doesn't exist, create it by following these steps:

- a. In SSMS, double-click **Security** to expand it.
- b. Right-click **Logins**, and then select **New login**.
- c. In the generated script with placeholders, edit and run the following SQL query:

```
CREATE LOGIN <SQL_login_name, sysname, login_name>
WITH PASSWORD = '<password, sysname, Change_Password>'
GO
```

5. Double-click **Database**.
6. Select the database that you want to grant the user permission to.
7. Double-click **Security**.
8. Right-click **Users**, and then select **New User**.
9. In the generated script with placeholders, edit and run the following SQL query:

```
CREATE USER <user_name, sysname, user_name>
FOR LOGIN <login_name, sysname, login_name>
WITH DEFAULT_SCHEMA = <default_schema, sysname, dbo>
GO
-- Add user to the database owner role

EXEC sp_addrolemember N'db_owner', N'<user_name, sysname, user_name>'
GO
```

#### NOTE

You can also use `sp_addrolemember` to map specific users to specific database roles.

For more information, see [Managing databases and logins in Azure SQL Database](#).

## Connection timeout expired errors

### System.Data.SqlClient.SqlException (0x80131904): Connection Timeout Expired

```
System.Data.SqlClient.SqlException (0x80131904): Connection Timeout Expired. The timeout period elapsed while attempting to consume the pre-login handshake acknowledgement. This could be because the pre-login handshake failed or the server was unable to respond back in time. The duration spent while attempting to connect to this server was - [Pre-Login] initialization=3; handshake=29995;
```

### System.Data.SqlClient.SqlException (0x80131904): Timeout expired

```
System.Data.SqlClient.SqlException (0x80131904): Timeout expired. The timeout period elapsed prior to completion of the operation or the server is not responding.
```

### System.Data.Entity.Core.EntityException: The underlying provider failed on Open

```
System.Data.Entity.Core.EntityException: The underlying provider failed on Open. ->
System.Data.SqlClient.SqlException: Timeout expired. The timeout period elapsed prior to completion of the operation or the server is not responding. -> System.ComponentModel.Win32Exception: The wait operation timed out
```

### Cannot connect to < server name >

```
Cannot connect to <server name>. ADDITIONAL INFORMATION: Connection Timeout Expired. The timeout period elapsed during the post-login phase. The connection could have timed out while waiting for server to complete the login process and respond; Or it could have timed out while attempting to create multiple active connections. The duration spent while attempting to connect to this server was - [Pre-Login] initialization=231; handshake=983; [Login] initialization=0; authentication=0; [Post-Login] complete=13000; (Microsoft SQL Server, Error: -2) For help, click: http://go.microsoft.com/fwlink?ProdName=Microsoft%20SQL%20Server&EvtSrc=MSSQLServer&EvtID=-2&LinkId=20476 The wait operation timed out
```

These exceptions can occur either because of connection or query issues. To confirm that this error is caused by connectivity issues, see [Confirm whether an error is caused by a connectivity issue](#).

Connection timeouts occur because the application can't connect to the server. To resolve this issue, try the steps

(in the order presented) in the [Steps to fix common connection issues](#) section.

## Resource governance errors

### Error 10928: Resource ID: %d

10928: Resource ID: %d. The %s limit for the database is %d and has been reached. See <http://go.microsoft.com/fwlink/?LinkId=267637> for assistance. The Resource ID value in error message indicates the resource for which limit has been reached. For sessions, Resource ID = 2.

To work around this issue, try one of the following methods:

- Verify whether there are long-running queries.

#### NOTE

This is a minimalist approach that might not resolve the issue.

1. Run the following SQL query to check the [sys.dm\\_exec\\_requests](#) view to see any blocking requests:

```
SELECT * FROM dm_exec_requests
```

2. Determine the **input buffer** for the head blocker.
3. Tune the head blocker query.

For an in-depth troubleshooting procedure, see [Is my query running fine in the cloud?](#).

If the database consistently reaches its limit despite addressing blocking and long-running queries, consider upgrading to an edition with more resources [Editions](#)).

For more information about dynamic management views, see [System dynamic management views](#).

For more information about database limits, see [SQL Database resource limits for Azure SQL Database server](#).

### Error 10929: Resource ID: 1

10929: Resource ID: 1. The %s minimum guarantee is %d, maximum limit is %d and the current usage for the database is %d. However, the server is currently too busy to support requests greater than %d for this database. See <http://go.microsoft.com/fwlink/?LinkId=267637> for assistance. Otherwise, please try again later.

### Error 40501: The service is currently busy

40501: The service is currently busy. Retry the request after 10 seconds. Incident ID: %ls. Code: %d.

This is an engine throttling error, an indication that resource limits are being exceeded.

For more information about resource limits, see [Database server resource limits](#).

### Error 40544: The database has reached its size quota

40544: The database has reached its size quota. Partition or delete data, drop indexes, or consult the documentation for possible resolutions. Incident ID: <ID>. Code: <code>.

This error occurs when the database has reached its size quota.

The following steps can either help you work around the problem or provide you with additional options:

1. Check the current size of the database by using the dashboard in the Azure portal.

#### NOTE

To identify which tables are consuming the most space and are therefore potential candidates for cleanup, run the following SQL query:

```
SELECT o.name,
 SUM(p.row_count) AS 'Row Count',
 SUM(p.reserved_page_count) * 8.0 / 1024 AS 'Table Size (MB)'
 FROM sys.objects o
 JOIN sys.dm_db_partition_stats p ON p.object_id = o.object_id
 GROUP BY o.name
 ORDER BY [Table Size (MB)] DESC
```

2. If the current size does not exceed the maximum size supported for your edition, you can use ALTER DATABASE to increase the MAXSIZE setting.
3. If the database is already past the maximum supported size for your edition, try one or more of the following steps:
  - Perform normal database cleanup activities. For example, clean up the unwanted data by using truncate/delete, or move data out by using SQL Server Integration Services (SSIS) or the bulk copy program (bcp) utility.
  - Partition or delete data, drop indexes, or consult the documentation for possible resolutions.
  - For database scaling, see [Scale single database resources](#) and [Scale elastic pool resources](#).

#### Error 40549: Session is terminated because you have a long-running transaction

```
40549: Session is terminated because you have a long-running transaction. Try shortening your transaction.
```

If you repeatedly encounter this error, try to resolve the issue by following these steps:

1. Check the sys.dm\_exec\_requests view to see any open sessions that have a high value for the total\_elapsed\_time column. Perform this check by running the following SQL script:

```
SELECT * FROM dm_exec_requests
```

2. Determine the input buffer for the long-running query.
3. Tune the query.

Also consider batching your queries. For information on batching, see [How to use batching to improve SQL Database application performance](#).

For an in-depth troubleshooting procedure, see [Is my query running fine in the cloud?](#).

#### Error 40551: The session has been terminated because of excessive TEMPDB usage

```
40551: The session has been terminated because of excessive TEMPDB usage. Try modifying your query to reduce the temporary table space usage.
```

To work around this issue, follow these steps:

1. Change the queries to reduce temporary table space usage.
2. Drop temporary objects after they're no longer needed.
3. Truncate tables or remove unused tables.

#### Error 40552: The session has been terminated because of excessive transaction log space usage

40552: The session has been terminated because of excessive transaction log space usage. Try modifying fewer rows in a single transaction.

To resolve this issue, try the following methods:

- The issue can occur because of insert, update, or delete operations. Try to reduce the number of rows that are operated on immediately by implementing batching or splitting into multiple smaller transactions.
- The issue can occur because of index rebuild operations. To work around this issue, make sure the number of rows that are affected in the table \* (average size of field that's updated in bytes + 80) < 2 gigabytes (GB).

**NOTE**

For an index rebuild, the average size of the field that's updated should be substituted by the average index size.

**Error 40553: The session has been terminated because of excessive memory usage**

40553 : The session has been terminated because of excessive memory usage. Try modifying your query to process fewer rows.

To work around this issue, try to optimize the query.

For an in-depth troubleshooting procedure, see [Is my query running fine in the cloud?](#).

**Table of additional resource governance error messages**

ERROR CODE	SEVERITY	DESCRIPTION
10928	20	<p>Resource ID: %d. The %s limit for the database is %d and has been reached. For more information, see <a href="#">SQL Database resource limits for single and pooled databases</a>.</p> <p>The Resource ID indicates the resource that has reached the limit. For worker threads, the Resource ID = 1. For sessions, the Resource ID = 2.</p> <p>For more information about this error and how to resolve it, see:</p> <ul style="list-style-type: none"><li>• <a href="#">Database server resource limits</a></li><li>• <a href="#">DTU-based limits for single databases</a></li><li>• <a href="#">DTU-based limits for elastic pools</a></li><li>• <a href="#">vCore-based limits for single databases</a></li><li>• <a href="#">vCore-based limits for elastic pools</a></li><li>• <a href="#">Managed instance resource limits</a>.</li></ul>

ERROR CODE	SEVERITY	DESCRIPTION
10929	20	<p>Resource ID: %d. The %s minimum guarantee is %d, maximum limit is %d, and the current usage for the database is %d. However, the server is currently too busy to support requests greater than %d for this database. The Resource ID indicates the resource that has reached the limit. For worker threads, the Resource ID = 1. For sessions, the Resource ID = 2. For more information, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">Database server resource limits</a></li> <li>• <a href="#">DTU-based limits for single databases</a></li> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for single databases</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a></li> <li>• <a href="#">Managed instance resource limits</a>.</li> </ul> <p>Otherwise, please try again later.</p>
40544	20	<p>The database has reached its size quota. Partition or delete data, drop indexes, or consult the documentation for possible resolutions. For database scaling, see <a href="#">Scale single database resources</a> and <a href="#">Scale elastic pool resources</a>.</p>
40549	16	<p>Session is terminated because you have a long-running transaction. Try shortening your transaction. For information on batching, see <a href="#">How to use batching to improve SQL Database application performance</a>.</p>
40550	16	<p>The session has been terminated because it has acquired too many locks. Try reading or modifying fewer rows in a single transaction. For information on batching, see <a href="#">How to use batching to improve SQL Database application performance</a>.</p>
40551	16	<p>The session has been terminated because of excessive <code>TEMPDB</code> usage. Try modifying your query to reduce the temporary table space usage.</p> <p>If you are using temporary objects, conserve space in the <code>TEMPDB</code> database by dropping temporary objects after they are no longer needed by the session. For more information on tempdb usage in SQL Database, see <a href="#">Tempdb database in SQL Database</a>.</p>

ERROR CODE	SEVERITY	DESCRIPTION
40552	16	<p>The session has been terminated because of excessive transaction log space usage. Try modifying fewer rows in a single transaction. For information on batching, see <a href="#">How to use batching to improve SQL Database application performance</a>.</p> <p>If you perform bulk inserts using the <code>bcp.exe</code> utility or the <code>System.Data.SqlClient.SqlBulkCopy</code> class, try using the <code>-b batchsize</code> or <code>BatchSize</code> options to limit the number of rows copied to the server in each transaction. If you are rebuilding an index with the <code>ALTER INDEX</code> statement, try using the <code>REBUILD WITH ONLINE = ON</code> option. For information on transaction log sizes for the vCore purchasing model, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">vCore-based limits for single databases</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a></li> <li>• <a href="#">Managed instance resource limits</a>.</li> </ul>
40553	16	<p>The session has been terminated because of excessive memory usage. Try modifying your query to process fewer rows.</p> <p>Reducing the number of <code>ORDER BY</code> and <code>GROUP BY</code> operations in your Transact-SQL code reduces the memory requirements of your query. For database scaling, see <a href="#">Scale single database resources</a> and <a href="#">Scale elastic pool resources</a>.</p>

## Elastic pool errors

The following errors are related to creating and using elastic pools:

ERROR CODE	SEVERITY	DESCRIPTION	CORRECTIVE ACTION
1132	17	<p>The elastic pool has reached its storage limit. The storage usage for the elastic pool cannot exceed (%d) MBs. Attempting to write data to a database when the storage limit of the elastic pool has been reached. For information on resource limits, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a>.</li> </ul>	<p>Consider increasing the DTUs of and/or adding storage to the elastic pool if possible in order to increase its storage limit, reduce the storage used by individual databases within the elastic pool, or remove databases from the elastic pool. For elastic pool scaling, see <a href="#">Scale elastic pool resources</a>.</p>

ERROR CODE	SEVERITY	DESCRIPTION	CORRECTIVE ACTION
10929	16	<p>The %s minimum guarantee is %d, maximum limit is %d, and the current usage for the database is %d.</p> <p>However, the server is currently too busy to support requests greater than %d for this database.</p> <p>For information on resource limits, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">DTU-based limits for elastic pools</a></li> <li>• <a href="#">vCore-based limits for elastic pools</a>.</li> </ul> <p>Otherwise, please try again later. DTU / vCore min per database; DTU / vCore max per database. The total number of concurrent workers (requests) across all databases in the elastic pool attempted to exceed the pool limit.</p>	<p>Consider increasing the DTUs or vCores of the elastic pool if possible in order to increase its worker limit, or remove databases from the elastic pool.</p>
40844	16	<p>Database '%ls' on Server '%ls' is a '%ls' edition database in an elastic pool and cannot have a continuous copy relationship.</p>	N/A
40857	16	<p>Elastic pool not found for server: '%ls', elastic pool name: '%ls'. Specified elastic pool does not exist in the specified server.</p>	<p>Provide a valid elastic pool name.</p>
40858	16	<p>Elastic pool '%ls' already exists in server: '%ls'. Specified elastic pool already exists in the specified SQL Database server.</p>	<p>Provide new elastic pool name.</p>
40859	16	<p>Elastic pool does not support service tier '%ls'. Specified service tier is not supported for elastic pool provisioning.</p>	<p>Provide the correct edition or leave service tier blank to use the default service tier.</p>
40860	16	<p>Elastic pool '%ls' and service objective '%ls' combination is invalid. Elastic pool and service tier can be specified together only if resource type is specified as 'ElasticPool'.</p>	<p>Specify correct combination of elastic pool and service tier.</p>

ERROR CODE	SEVERITY	DESCRIPTION	CORRECTIVE ACTION
40861	16	The database edition '%.*ls' cannot be different than the elastic pool service tier which is '%.*ls'. The database edition is different than the elastic pool service tier.	Do not specify a database edition which is different than the elastic pool service tier. Note that the database edition does not need to be specified.
40862	16	Elastic pool name must be specified if the elastic pool service objective is specified. Elastic pool service objective does not uniquely identify an elastic pool.	Specify the elastic pool name if using the elastic pool service objective.
40864	16	The DTUs for the elastic pool must be at least (%d) DTUs for service tier '%.*ls'. Attempting to set the DTUs for the elastic pool below the minimum limit.	Retry setting the DTUs for the elastic pool to at least the minimum limit.
40865	16	The DTUs for the elastic pool cannot exceed (%d) DTUs for service tier '%.*ls'. Attempting to set the DTUs for the elastic pool above the maximum limit.	Retry setting the DTUs for the elastic pool to no greater than the maximum limit.
40867	16	The DTU max per database must be at least (%d) for service tier '%.*ls'. Attempting to set the DTU max per database below the supported limit.	Consider using the elastic pool service tier that supports the desired setting.
40868	16	The DTU max per database cannot exceed (%d) for service tier '%.*ls'. Attempting to set the DTU max per database beyond the supported limit.	Consider using the elastic pool service tier that supports the desired setting.
40870	16	The DTU min per database cannot exceed (%d) for service tier '%.*ls'. Attempting to set the DTU min per database beyond the supported limit.	Consider using the elastic pool service tier that supports the desired setting.
40873	16	The number of databases (%d) and DTU min per database (%d) cannot exceed the DTUs of the elastic pool (%d). Attempting to specify DTU min for databases in the elastic pool that exceeds the DTUs of the elastic pool.	Consider increasing the DTUs of the elastic pool, or decrease the DTU min per database, or decrease the number of databases in the elastic pool.

ERROR CODE	SEVERITY	DESCRIPTION	CORRECTIVE ACTION
40877	16	An elastic pool cannot be deleted unless it does not contain any databases. The elastic pool contains one or more databases and therefore cannot be deleted.	Remove databases from the elastic pool in order to delete it.
40881	16	The elastic pool '%.*ls' has reached its database count limit. The database count limit for the elastic pool cannot exceed (%d) for an elastic pool with (%d) DTUs. Attempting to create or add database to elastic pool when the database count limit of the elastic pool has been reached.	Consider increasing the DTUs of the elastic pool if possible in order to increase its database limit, or remove databases from the elastic pool.
40889	16	The DTUs or storage limit for the elastic pool '%.*ls' cannot be decreased since that would not provide sufficient storage space for its databases. Attempting to decrease the storage limit of the elastic pool below its storage usage.	Consider reducing the storage usage of individual databases in the elastic pool or remove databases from the pool in order to reduce its DTUs or storage limit.
40891	16	The DTU min per database (%d) cannot exceed the DTU max per database (%d). Attempting to set the DTU min per database higher than the DTU max per database.	Ensure the DTU min per databases does not exceed the DTU max per database.
TBD	16	The storage size for an individual database in an elastic pool cannot exceed the max size allowed by '%.*ls' service tier elastic pool. The max size for the database exceeds the max size allowed by the elastic pool service tier.	Set the max size of the database within the limits of the max size allowed by the elastic pool service tier.

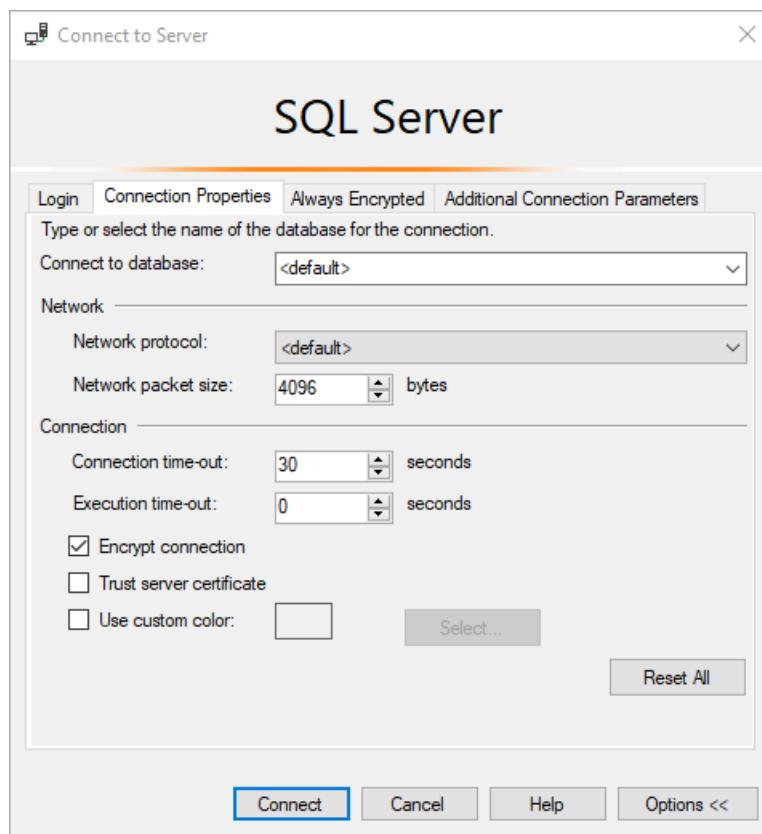
## Cannot open database "master" requested by the login. The login failed

This issue occurs because the account doesn't have permission to access the master database. But by default, SQL Server Management Studio (SSMS) tries to connect to the master database.

To resolve this issue, follow these steps:

1. On the login screen of SSMS, select **Options**, and then select **Connection Properties**.
2. In the **Connect to database** field, enter the user's default database name as the default login database,

and then select **Connect**.



## Confirm whether an error is caused by a connectivity issue

To confirm whether an error is caused by a connectivity issue, review the stack trace for frames that show calls to open a connection like the following ones (note the reference to the **SqlConnection** class):

```
System.Data.SqlClient.SqlConnection.TryOpen(TaskCompletionSource`1 retry)
at System.Data.SqlClient.SqlConnection.Open()
at AzureConnectionTest.Program.Main(String[] args)
ClientConnectionId:<Client connection ID>
```

When the exception is triggered by query issues, you'll notice a call stack that's similar to the following (note the reference to the **SqlCommand** class). In this situation, [tune your queries](#).

```
at System.Data.SqlClient.SqlCommand.ExecuteReader()
at AzureConnectionTest.Program.Main(String[] args)
ClientConnectionId:<Client ID>
```

For additional guidance on fine-tuning performance, see the following resources:

- [How to maintain Azure SQL indexes and statistics](#)
- [Manual tune query performance in Azure SQL Database](#)
- [Monitoring performance Azure SQL Database by using dynamic management views](#)
- [Operating the Query Store in Azure SQL Database](#)

## Steps to fix common connection issues

1. Make sure that TCP/IP is enabled as a client protocol on the application server. For more information, see [Configure client protocols](#). On application servers where you don't have SQL Server tools installed, verify that TCP/IP is enabled by running **cliconfig.exe** (SQL Server Client Network utility).

2. Check the application's connection string to make sure it's configured correctly. For example, make sure that the connection string specifies the correct port (1433) and fully qualified server name. See [Get SQL Server connection information](#).
3. Try increasing the connection timeout value. We recommend using a connection timeout of at least 30 seconds.
4. Test the connectivity between the application server and the Azure SQL database by using [SQL Server management Studio \(SSMS\)](#), a UDL file, ping, or telnet. For more information, see [Troubleshooting SQL Server connectivity issues](#) and [Diagnostics for connectivity issues](#).

**NOTE**

As a troubleshooting step, you can also test connectivity on a different client computer.

5. As a best practice, make sure that the retry logic is in place. For more information about retry logic, see [Troubleshoot transient faults and connection errors to SQL Database](#).

If these steps don't resolve your problem, try to collect more data and then contact support. If your application is a cloud service, enable logging. This step returns a UTC time stamp of the failure. Additionally, SQL Azure returns the tracing ID. [Microsoft Customer Support Services](#) can use this information.

For more information about how to enable logging, see [Enable diagnostics logging for apps in Azure App Service](#).

## Next steps

- [Azure SQL connectivity architecture](#)
- [Azure SQL Database and Data Warehouse network access controls](#)

# Azure SQL Database Import/Export service takes a long time to import or export a database

2/21/2020 • 2 minutes to read • [Edit Online](#)

When you use the Azure SQL Database Import/Export service, the process might take longer than expected. This article describes the potential causes for this delay and alternative workaround methods.

## Azure SQL Database Import/Export service

The Azure SQL Database Import/Export service is a REST-based web service that runs in every Azure data center. This service is called when you use either the [Import database](#) or [Export](#) option to move your SQL database in the Azure portal. The service provides free request queuing and compute services to perform imports and exports between an Azure SQL database and Azure Blob storage.

The import and export operations don't represent a traditional physical database backup but instead a logical backup of the database that uses a special BACPAC format. The BACPAC format lets you avoid having to use a physical format that might vary between versions of Microsoft SQL Server and Azure SQL Database. Therefore, you can use it to safely restore the database to a SQL Server database and to a SQL database.

## What causes delays in the process?

The Azure SQL Database Import/Export service provides a limited number of compute virtual machines (VMs) per region to process import and export operations. The compute VMs are hosted per region to make sure that the import or export avoids cross-region bandwidth delays and charges. If too many requests are made at the same time in the same region, significant delays can occur in processing the operations. The time that's required to complete requests can vary from a few seconds to many hours.

### NOTE

If a request is not processed within four days, the service automatically cancels the request.

## Recommended solutions

If your database exports are used only for recovery from accidental data deletion, all the Azure SQL Database editions provide self-service restoration capability from system-generated backups. But if you need these exports for other reasons, and if you require consistently faster or more predictable import/export performance, consider the following options:

- [Export to a BACPAC file by using the SQLPackage utility.](#)
- [Export to a BACPAC file by using SQL Server Management Studio \(SSMS\).](#)
- Run the BACPAC import or export directly in your code by using the Microsoft SQL Server Data-Tier Application Framework (DacFx) API. For additional information, see:
  - [Export a data-tier application](#)
  - [Microsoft.SqlServer.Dac Namespace](#)
  - [Download DACFx](#)

## Things to consider when you export or import an Azure SQL database

- All the methods discussed in this article use up the Database Transaction Unit (DTU) quota, which causes throttling by the Azure SQL Database service. You can [view the DTU stats for the database on the Azure portal](#). If the database has reached its resource limits, [upgrade the service tier](#) to add more resources.
- Ideally, you should run client applications (like the sqlpackage utility or your custom DAC application) from a VM in the same region as your SQL database. Otherwise, you might experience performance issues related to network latency.
- Exporting large tables without clustered indexes can be very slow or even cause failure. This behavior occurs because the table can't be split up and exported in parallel. Instead, it must be exported in a single transaction, and that causes slow performance and potential failure during export, especially for large tables.

## Related documents

[Considerations when exporting an Azure SQL database](#)