

# Contents

[SQL DB Documentation](#)

[SQL Database Service](#)

[About SQL DB](#)

[Choose a version of Azure SQL](#)

[Quickstarts](#)

[Create](#)

[Single database](#)

[Managed Instance](#)

[Configure](#)

[Logical server firewall rule](#)

[Client VM connection to a Managed Instance](#)

[Point-to-site connection to a Managed Instance](#)

[Load data](#)

[Import BACPAC into a single database](#)

[Restore a database backup to a Managed Instance](#)

[Query](#)

[SSMS](#)

[Azure Data Studio](#)

[Azure portal](#)

[VS Code](#)

[.NET with Visual Studio](#)

[.NET core](#)

[.NET with Active Directory MFA](#)

[Go](#)

[Java](#)

[Node.js](#)

[PHP](#)

[Python](#)

[Ruby](#)

## Tutorials

- 1 - Design DB using SSMS
- 2 - Design DB using .NET
- 3 - Migrate to a single Azure SQL database using DMS
- 4 - Secure your SQL DB
- 5 - Implement a geo-distributed database
- 6 - Migrate to a Managed Instance using DMS
- 7 - Set up Azure SQL Data Sync

## Samples

- Code samples
- Azure CLI
- Azure PowerShell

## Concepts

- Purchasing models
  - vCore vs. DTU
  - vCore-based purchasing model
    - Choosing a vCore service tier
    - General Purpose / Business Critical
  - Hyperscale
  - Hyperscale FAQ
  - DTU service tiers
  - Prepay for reserved capacity
- Logical servers
  - Create and manage
  - Resource limits
  - Connectivity architecture
  - T-SQL differences
  - Transactional replication - single DBs and elastic pools

## Single databases

- Create and manage
- vCore-based resource limits
- DTU-based resource limits

[Scale resources](#)

[Elastic pools](#)

[Elastic pools](#)

[Create and manage](#)

[vCore-based resource limits](#)

[DTU-based resource limits](#)

[Scale resources](#)

[Managed instances](#)

[Managed instances](#)

[Resource limits](#)

[VNet configuration](#)

[Custom DNS](#)

[Sync network configuration](#)

[Connectivity architecture](#)

[Connect applications](#)

[T-SQL differences](#)

[Transactional replication](#)

[Business continuity](#)

[Business continuity](#)

[High availability](#)

[Accelerated database recovery](#)

[Database backups](#)

[Long-term backup retention](#)

[Long-term backup retention](#)

[Configure using Azure blob storage](#)

[Configure using Azure vault \(deprecated\)](#)

[Failover groups and geo-replication](#)

[Failover groups and geo-replication](#)

[Configure geo-replication - Portal](#)

[Configure security for replicas](#)

[Database recovery options](#)

[Recovery using backups](#)

- [Outage recovery guidance](#)
- [Recovery drills](#)
- [App design](#)
  - [Design for disaster recovery](#)
  - [Design for elastic pools](#)
  - [Design for app upgrades](#)
- [Security](#)
  - [Security](#)
    - [Advanced Threat Protection](#)
      - [Advanced Threat Protection](#)
      - [Data discovery and classification](#)
      - [Vulnerability assessment](#)
      - [Threat detection - SQL Database](#)
      - [Threat detection - Managed Instance](#)
    - [Firewall rules](#)
      - [Create and manage](#)
      - [Virtual network endpoints](#)
      - [Virtual network endpoints - PowerShell](#)
    - [Access control](#)
      - [Access control](#)
      - [Logins and users](#)
    - [Azure AD authentication](#)
      - [Azure AD authentication](#)
      - [Configure Azure AD auth](#)
      - [Conditional access](#)
      - [Multi-factor auth](#)
      - [Configure multi-factor auth](#)
    - [Auditing](#)
      - [Auditing - Servers, pools, and databases](#)
      - [Auditing - Managed Instance](#)
      - [Table auditing \(deprecated\)](#)
    - [Dynamic data masking](#)

[Dynamic data masking](#)

[Configure masking](#)

[Always encrypted](#)

[Use the certificate store](#)

[Use the Azure key vault](#)

[TDE with Azure SQL](#)

[TDE with Azure SQL](#)

[TDE with Bring Your Own Key](#)

[Configure TDE with BYOK](#)

[Rotate TDE BYOK keys](#)

[Remove TDE protector](#)

[Elastic scalability](#)

[Resource scalability](#)

[Read Scale-Out](#)

[Database sharding](#)

[Database sharding](#)

[Elastic client library - consolidate](#)

[Upgrade elastic database client library](#)

[Shard maps](#)

[Query routing](#)

[Manage credentials](#)

[Shard querying](#)

[Elastic tools](#)

[Move sharded data](#)

[Elastic client library - Dapper](#)

[Elastic tools FAQ](#)

[Create sharded app](#)

[Query horizontally-sharded data](#)

[Move sharded data](#)

[Security configuration](#)

[Add a shard](#)

[Fix shard map problems](#)

- [Migrate sharded DB](#)
- [Create counters](#)
- [Use entity framework](#)
- [SQL features comparison](#)
- [Monitoring and performance tuning](#)
  - [Monitoring and tuning overview](#)
  - [Intelligent performance](#)
    - [Tuning advisors](#)
    - [Tuning actions and recommendations](#)
    - [Query Performance Insight](#)
    - [Database advisor performance recommendations](#)
    - [Apply performance recommendations](#)
  - [Advanced monitoring](#)
    - [Azure SQL Analytics monitoring](#)
    - [Diagnostic telemetry logging](#)
  - [Automated troubleshooting](#)
    - [Intelligent insights into performance](#)
    - [Troubleshoot performance with Intelligent Insights](#)
    - [Use Intelligent Insights diagnostics log](#)
  - [Automatic tuning](#)
    - [Automatic tuning overview](#)
    - [Enable automatic tuning](#)
    - [Enable E-mail notifications](#)
  - [Intelligent query processing](#)
    - [Intelligent query processing](#)
    - [Adaptive query processing](#)
  - [Manual tuning](#)
    - [Manual tuning](#)
  - [Use DMVs to monitor performance](#)
  - [Operate query store](#)
  - [In-memory](#)
  - [Extended events](#)

- [Extended events](#)
- [Extended events - event file](#)
- [Extended events - ring buffer](#)
- [Create alerts](#)
- [Troubleshoot connectivity](#)
- [Azure Automation](#)
- [Cross-database operations](#)
  - [Elastic Database jobs](#)
    - [Elastic Database jobs](#)
    - [Create and manage \(PowerShell\)](#)
    - [Create and manage \(T-SQL\)](#)
    - [Migrate \(from old Elastic jobs\)](#)
    - [Elastic jobs \(deprecated\)](#)
      - [Elastic jobs \(deprecated\)](#)
      - [Install elastic job \(deprecated\)](#)
      - [Create job - Portal \(deprecated\)](#)
      - [Create job - Azure PowerShell \(deprecated\)](#)
      - [Create job - Sample app \(deprecated\)](#)
      - [Uninstall elastic job \(deprecated\)](#)
    - [Elastic queries](#)
      - [Elastic queries](#)
      - [Query vertically partitioned data](#)
      - [Report across scaled-out data tier](#)
      - [Query across tables with different schemas](#)
    - [Elastic transactions](#)
  - [Multi-tenant SaaS](#)
    - [SaaS design patterns](#)
    - [SaaS video indexer](#)
    - [SaaS app security](#)
  - [Develop databases](#)
    - [Develop databases](#)
    - [JSON data](#)

- [In-memory](#)
- [Temporal tables](#)
- [Retention policies](#)
- [Configure In-Memory](#)
- [How-to guides](#)
  - [Migrate to SQL Database](#)
  - [Migrate to SQL Database](#)
  - [Migrate to Managed Instance](#)
  - [Migrate TDE cert to Managed Instance](#)
  - [Manage SQL Database after migration](#)
- [Load and move data](#)
  - [Copy a DB within Azure](#)
  - [Import a DB from a BACPAC](#)
  - [Export a DB to BACPAC](#)
- [Data sync](#)
  - [SQL Data Sync](#)
  - [Replicate schema changes](#)
  - [Monitor with OMS](#)
  - [Best practices for Data Sync](#)
  - [Troubleshoot Data Sync](#)
- [Load data with BCP](#)
- [Load data with ADF](#)
- [Manage file space](#)
- [Develop apps](#)
  - [Connectivity](#)
  - [Use Spark Connector](#)
  - [Authenticate app](#)
  - [Error messages](#)
  - [Batching for perf](#)
  - [Connectivity guidance](#)
  - [DNS aliases](#)
  - [DNS alias PowerShell](#)

- [Ports - ADO.NET](#)
- [C and C ++](#)
- [Excel](#)
- [Multi-tenant SaaS sample application](#)
  - [Wingtip Tickets sample](#)
  - [General guidance](#)
  - [Standalone application](#)
    - [Deploy example app](#)
    - [Provision tenants](#)
  - [Database per tenant](#)
    - [Tutorial intro](#)
    - [Deploy example app](#)
    - [Provision tenants](#)
    - [Monitor database performance](#)
    - [Monitor with log analytics](#)
    - [Restore one tenant](#)
    - [Manage tenant schema](#)
    - [Cross-tenant reporting](#)
    - [Tenant analytics](#)
      - [With SQL DB](#)
      - [With SQL DW](#)
    - [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](#)
- [Reference](#)
  - [Azure PowerShell](#)

[Azure CLI](#)

[.NET](#)

[Java](#)

[Node.js](#)

[Python](#)

[Ruby](#)

[PHP](#)

[T-SQL](#)

[REST](#)

[SQL Server tools](#)

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

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

[BCP](#)

[SQLCMD](#)

[SqlPackage](#)

[SQL Database Management Library package](#)

[SQL Server drivers](#)

[ADO.NET](#)

[JDBC](#)

[ODBC](#)

[Resources](#)

[Build your skills with Microsoft Learn](#)

[Azure Roadmap](#)

[FAQ](#)

[Public data sets](#)

[Troubleshoot connectivity](#)

[Pricing](#)

[MSDN forum](#)

[Stack Overflow](#)

[Videos](#)

[Service updates](#)

[Architecture center](#)

## Customer stories

# The Azure SQL Database service

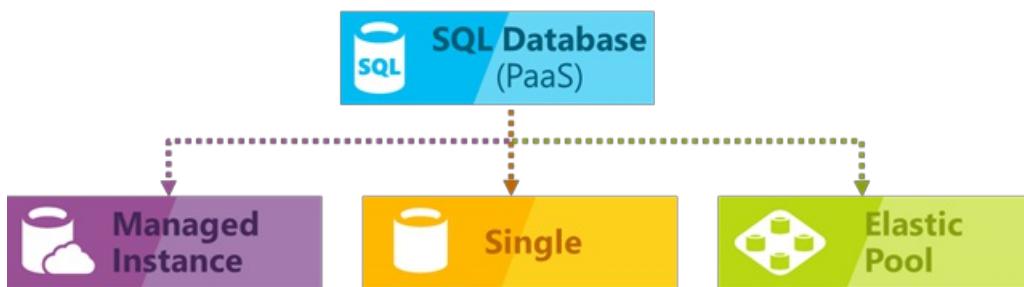
10/18/2018 • 14 minutes to read • [Edit Online](#)

SQL Database is a general-purpose relational database managed service in Microsoft Azure that supports structures such as relational data, JSON, spatial, and XML. SQL Database delivers dynamically scalable performance within two different purchasing models: a [vCore-based purchasing model](#) and a [DTU-based purchasing model](#). SQL Database also provides options such as [columnstore indexes](#) for extreme analytic analysis and reporting, and [in-memory OLTP](#) for extreme transactional processing. Microsoft handles all patching and updating of the SQL code base seamlessly and abstracts away all management of the underlying infrastructure.

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

- As a single database with its own set of resources managed via a logical server
- As a pooled database in an [elastic pool](#) with a shared set of resources managed via a logical server
- As a part of a collection of databases known as a [managed instance](#) that contains system and user databases and sharing a set of resources

The following illustration shows these deployment options:



SQL Database shares its code base with the [Microsoft SQL Server database engine](#). With Microsoft's cloud-first strategy, the newest capabilities of SQL Server are released first to SQL Database, and then to SQL Server itself. This approach provides you with the newest SQL Server capabilities with no overhead for patching or upgrading - and with these new features tested across millions of databases. For information about new capabilities as they are announced, see:

- [Azure Roadmap for SQL Database](#):

A place to find out what's new and what's coming next.

- [Azure SQL Database blog](#):

A place where SQL Server product team members blog about SQL Database news and features.

## IMPORTANT

To understand the feature differences between SQL Database and SQL Server, see [SQL features](#).

SQL Database delivers predictable performance with multiple resource types, service tiers, and compute sizes that provides dynamic scalability with no downtime, built-in intelligent optimization, global scalability and availability, and advanced security options — all with near-zero administration. These capabilities allow you to focus on rapid app development and accelerating your time to market, rather than allocating precious time and resources to managing virtual machines and infrastructure. The SQL Database service is currently in 38 data

centers around the world, with more data centers coming online regularly, which enables you to run your database in a data center near you.

## Scalable performance and pools

With SQL 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. SQL Database provides different compute sizes for different needs, and enables databases to be pooled to maximize the use of resources and save money.

- With [SQL Database Managed Instance](#), each instance is isolated from other instances with guaranteed resources. For more information, see [SQL Database Managed Instance](#).
- With the [Hyperscale service tier](#) (preview) in the vCore purchasing model, you can scale to 100 TB with fast backup and restore capabilities.

### Adjust performance and scale without downtime

SQL Database offers a [DTU-based purchasing model](#) or 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. Compute sizes 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 of memory, and the amount and speed of storage.

You can build your first app on a small, single database at a low cost per month in the General Purpose service tier and 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 and enables you to only pay for the resources that you need when you need them.

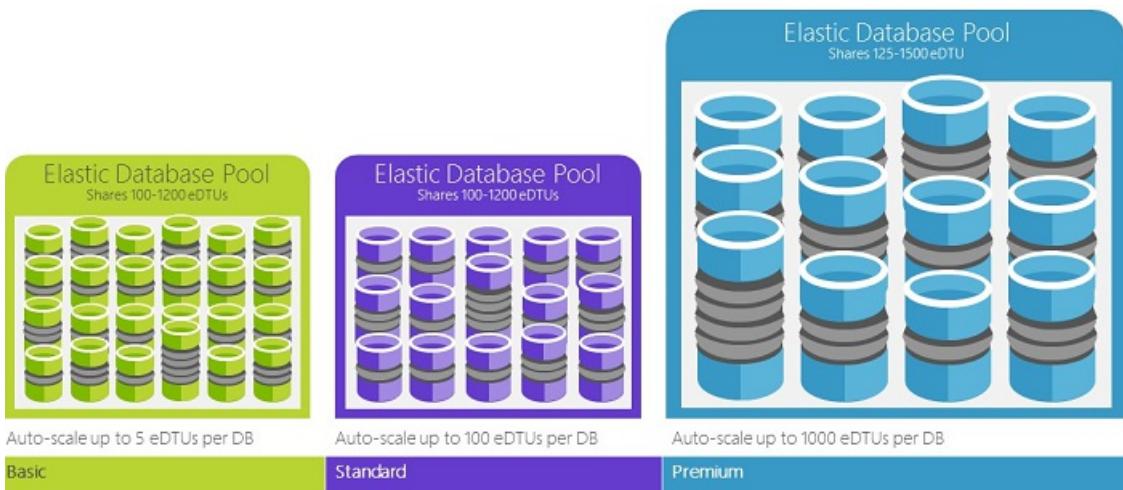
#### IMPORTANT

The [Hyperscale service tier](#) is currently in public preview. We don't recommend running any production workload in Hyperscale databases yet. You can't update a Hyperscale database to other service tiers. For test purpose, we recommend you make a copy of your current database and update the copy to Hyperscale service tier.

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. 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](#).

### 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. 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.



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. What's more, 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 SaaS applications using 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

SQL Database Managed Instance does not support elastic pools.

#### Blend single databases with pooled databases

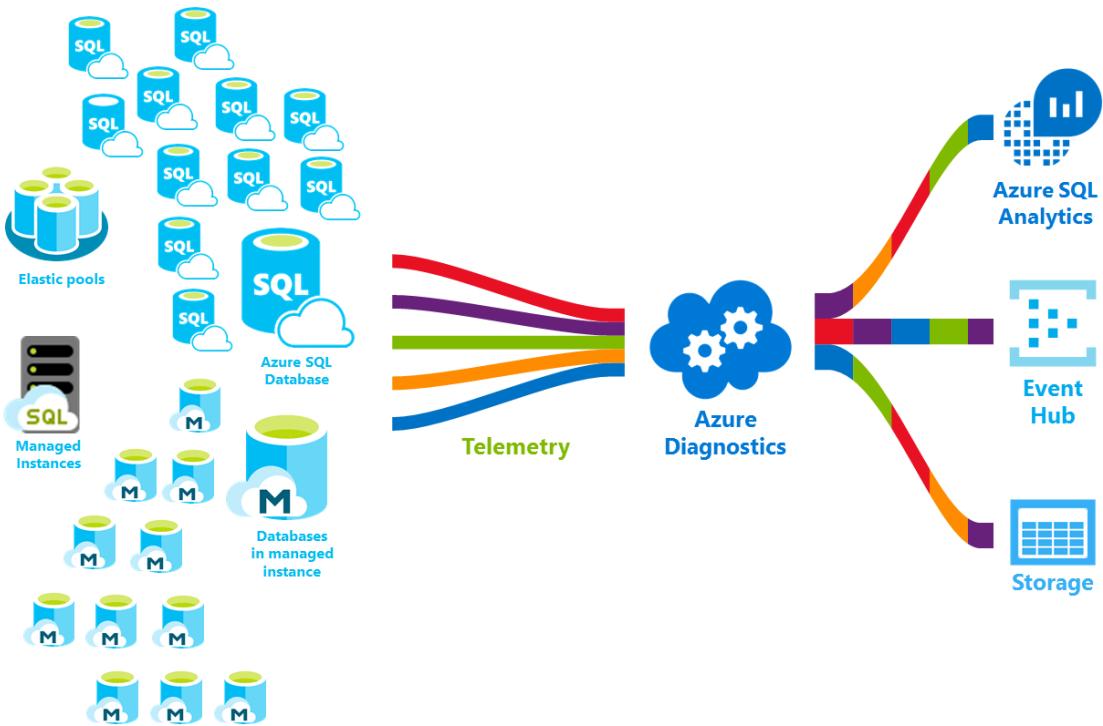
Either way you go — single databases or elastic pools — you are not locked in. You can blend single databases with elastic pools and change the service tiers of single databases and elastic pools quickly and easily to adapt to your situation. With the power and reach of Azure, you can mix-and-match other 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

But how can you compare the relative performance of single databases and elastic pools? How do you know the right click-stop when you dial up and down? 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. See [DTU-based purchasing model](#) and [vCore-based purchasing model](#) for details.

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 Hub:** For integrating SQL Database telemetry with your custom monitoring solution or hot pipelines
- **Azure Log Analytics:** For built-in monitoring solution with reporting, alerting, and mitigating capabilities.



## Availability capabilities

Azure's industry leading 99.99% availability service level agreement ([SLA](#)), powered by a global network of Microsoft-managed datacenters, helps keep your app running 24/7. The Azure platform fully manages every Azure SQL Database and guarantees no data loss and 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 under the covers. 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, including:

- **Automatic backups:**

SQL Database automatically performs full, differential, and transaction log backups.

- **Point-in-time restores:**

SQL Database supports recovery to any point in time within the automatic backup retention period.

- **Active geo-replication:**

SQL Database allows you to configure up to four readable secondary databases in either the same or globally distributed Azure data centers. 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 and remove bottlenecks on the primary that are due to read workloads.

- **Failover groups:**

SQL Database allows you to enable high availability and load balancing at global scale, including transparent geo-replication and failover of large sets of databases and elastic pools. Failover groups and active geo-replication enables creation of globally distributed SaaS applications with minimal administration overhead leaving 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, including 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 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 behind the scenes. Various algorithms are continuously evaluating the telemetry data so that the service can learn and adapt with your application. Based on this analysis, the service comes up with performance improving recommendations tailored to your specific workload.

### Automatic performance monitoring and tuning

SQL Database provides detailed insight into the queries that you need to monitor. SQL Database's 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 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 and it informs you of performance degradation issues, it identifies the root cause of the 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 portal provide. Instead of monitoring and tuning your database manually, you might consider delegating some of the monitoring and tuning actions to SQL Database 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 controlled and safe way. Automatic tuning means that the performance of your database is carefully monitored and compared before and after every tuning action, and if the performance doesn't improve, the tuning action is reverted.

Today, many of our partners running [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, since part of their customer base also uses SQL Server, they are using the same indexing recommendations provided by SQL Database to help their SQL Server customers.

There are two automatic tuning aspects that 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 (coming soon, already available in SQL Server 2017).

### Adaptive query processing

We are also adding the [adaptive query processing](#) family of features to SQL Database, 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.

## Advance Threat Protection

SQL Advanced Threat Protection 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 could indicate a threat to your database. It provides a single go-to location for enabling and managing these capabilities.

- [Data Discovery & Classification](#):

This feature (currently in preview) 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](#):

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 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. 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

[SQL Database 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 could indicate business concerns or suspected security violations.

## Data encryption

SQL Database secures your data by providing encryption for data in motion with [Transport Layer Security](#), for data at rest with [Transparent Data Encryption](#), and for data in use with [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](#) (MFA) to increase data and application security while supporting a single sign-in process.

## Compliance certification

SQL Database participates in regular audits and has been certified against several compliance standards. For more information, see the [Microsoft Azure Trust Center](#), where you can find the most current list of [SQL Database compliance certifications](#).

## 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 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, Azure 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 that supports extensions, including the [mssql extension](#) for querying Microsoft SQL Server, Azure SQL Database, and SQL Data Warehouse.

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

## 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 single database and elastic pools cost comparisons and calculators.
- See these quickstarts to get you 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](#)

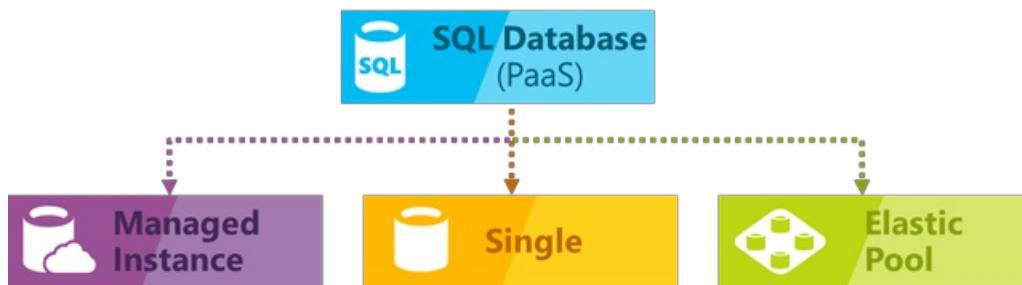
# Choose a cloud SQL Server option: Azure SQL (PaaS) Database or SQL Server on Azure VMs (IaaS)

10/16/2018 • 14 minutes to read • [Edit Online](#)

In Azure, you can have your SQL Server workloads running in a hosted infrastructure (IaaS) or running as a hosted service (PaaS):

- [Azure SQL Database](#): A SQL database engine, based on the Enterprise Edition of SQL Server, that is optimized for modern application development. Azure SQL Database offers several deployment options:
  - You can deploy a single database to a [logical server](#).
  - You can deploy into an [elastic pool](#) on a [logical server](#) to share resources and reduce costs.
  - You can deploy to a [Azure SQL Database Managed Instances](#).

The following illustration shows these deployment options:



## NOTE

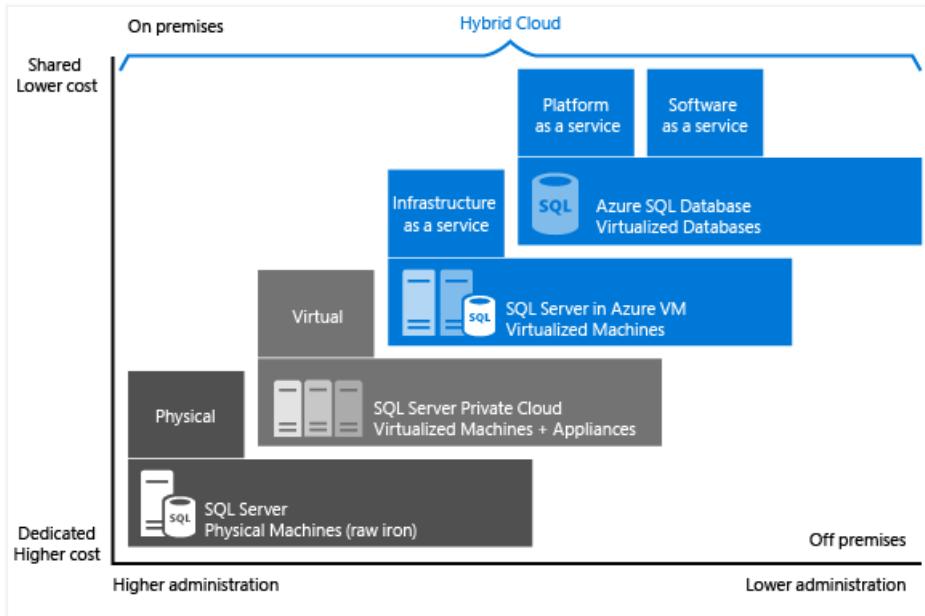
With all three versions, Azure SQL Database adds additional features that are not available in SQL Server, such as built-in intelligence and management. A logical server containing single and pooled databases offers most of database-scoped features of SQL Server. With Azure SQL Database Managed Instance, Azure SQL Database offers shared resources for databases and additional instance-scoped features. Azure SQL Database Managed Instance supports database migration with minimal to no database change.

- [SQL Server on Azure Virtual Machines](#): SQL Server installed and hosted in the cloud on Windows Server or Linux virtual machines (VMs) running on Azure, also known as an infrastructure as a service (IaaS). SQL Server on Azure 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 Database is that SQL Server VMs allow full control over the database engine. You can choose when maintenance/patching will start, to change the recovery model to simple or bulk logged to enable faster load less log, to pause or start engine when needed, and you can fully customize the SQL Server database engine. With this additional control comes with added responsibility to manage the virtual machines.

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

## Microsoft's SQL data platform

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 public cloud. SQL Server on Azure 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 (on the X axis), and by the degree of cost efficiency achieved by database level consolidation and automation (on the Y axis).

When designing an application, four basic options are available for hosting the SQL Server part of the application:

- SQL Server on non-virtualized physical machines
- SQL Server in on-premises virtualized machines (private cloud)
- SQL Server in Azure Virtual Machine (Microsoft public cloud)
- Azure SQL Database (Microsoft public cloud)

In the following sections, you learn about SQL Server in the Microsoft public cloud: Azure SQL Database and SQL Server on Azure VMs. In addition, you explore common business motivators for determining which option works best for your application.

## A closer look at Azure SQL Database and SQL Server on Azure VMs

- **Azure SQL Database**

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](#) is built on standardized hardware and software that is owned, hosted, and maintained by Microsoft. With SQL Database, you can use built-in features and functionality that require extensive configuration in SQL Server. When using SQL Database, you pay-as-you-go with options to scale up or out for greater power with no interruption. Azure SQL Database is an ideal environment for developing new applications in the cloud. And, with [Azure SQL Database Managed Instance](#), you can bring your own license.

Additionally, 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 Instance](#) is ideal for on-premises database migrations to Azure with minimal changes required.

- **SQL Server on Azure Virtual Machines (VMs)**

Falls into the industry category *Infrastructure-as-a-Service (IaaS)* and allows you to run SQL Server inside a virtual machine in the cloud. [SQL Server virtual machines](#) also run on standardized hardware that is owned, hosted, and maintained by Microsoft. When using SQL Server on a VM, 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.

In general, these two SQL options are optimized for different purposes:

- **Azure SQL Database**

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. [Azure SQL Database Managed Instance](#) provides support for instance-scoped features enabling easy migration of existing applications, as well as sharing resources amongst databases.

- **SQL Server running on Azure VMs**

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 Server on Azure VMs, 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.

The following table summarizes the main characteristics of SQL Database and SQL Server on Azure VMs:

|                  | AZURE SQL DATABASE<br>LOGICAL SERVERS, ELASTIC<br>POOLS, AND SINGLE<br>DATABASES   | AZURE SQL DATABASE<br>MANAGED INSTANCE  | AZURE VIRTUAL MACHINE<br>SQL SERVER   |
|------------------|--|---|---|
| <b>Best for:</b> | New cloud-designed applications that want to use the latest stable SQL Server features and have time constraints in development and marketing. | 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. | Existing applications that require fast migration to the cloud with minimal changes or no changes. Rapid development and test scenarios when you do not want to buy on-premises non-production SQL Server hardware. |
|                  | Teams that need built-in high availability, disaster recovery, and upgrade for the database.   | Same as SQL Database.   | Teams that can configure, fine tune, customize, and manage high availability, disaster recovery, and patching for SQL Server. Some provided automated features dramatically simplify this.                          |
|                  | Teams that do not want to manage the underlying operating system and configuration settings.   | Same as SQL Database.   | You need a customized environment with full administrative rights.  |

|                                 | AZURE SQL DATABASE<br>LOGICAL SERVERS, ELASTIC<br>POOLS, AND SINGLE<br>DATABASES  | AZURE SQL DATABASE<br>MANAGED INSTANCE   | AZURE VIRTUAL MACHINE<br>SQL SERVER   |
|---------------------------------|---|--|---|
|                                 | Databases of up to 100 TB.  | Same as SQL Database.  | SQL Server instances with up to 64 TB of storage. The instance can support as many databases as needed.   |
| <b>Compatibility</b>            | Supports most on-premises database-level capabilities.  | Supports almost all on-premises instance-level and database-level capabilities.  | Supports all on-premises capabilities.  |
| <b>Resources:</b>               | You do not want to employ IT resources for configuration and management of the underlying infrastructure but want to focus on the application layer.  | Same as SQL Database.  | You have some IT resources for configuration and management. Some provided automated features dramatically simplify this.   |
| <b>Total cost of ownership:</b> | Eliminates hardware costs and reduces administrative costs.   | Same as SQL Database.  | Eliminates hardware costs.  |
| <b>Business continuity:</b>     | In addition to <a href="#">built-in fault tolerance infrastructure capabilities</a> , Azure SQL Database provides features, such as <a href="#">automated backups</a> , <a href="#">Point-In-Time Restore</a> , <a href="#">geo-restore</a> , and <a href="#">failover groups and active geo-replication</a> to increase business continuity. For more information, see <a href="#">SQL Database business continuity overview</a> . | Same as SQL Database, plus user-initiated, copy-only backups are available.  | SQL Server on Azure VMs lets you set up a high availability and disaster recovery solution for your database's specific needs. Therefore, you can have a system that is highly optimized for your application. You can test and run failovers by yourself when needed. For more information, see <a href="#">High Availability and Disaster Recovery for SQL Server on Azure Virtual Machines</a> .   |
| <b>Hybrid cloud:</b>            | Your on-premises application can access data in Azure SQL Database.   | <a href="#">Native virtual network implementation</a> and connectivity to your on-premises environment using Azure Express Route or VPN Gateway. | With SQL Server on Azure VMs, 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 <a href="#">Azure Virtual Network</a> . In addition, you can store on-premises data files in Azure Storage using <a href="#">SQL Server Data Files in Azure</a> . For more information, see <a href="#">Introduction to SQL Server 2014 Hybrid Cloud</a> . |

|  | AZURE SQL DATABASE<br>LOGICAL SERVERS, ELASTIC<br>POOLS, AND SINGLE<br>DATABASES                 | AZURE SQL DATABASE<br>MANAGED INSTANCE                                | AZURE VIRTUAL MACHINE<br>SQL SERVER  |
|--|--|---|--|
|  | Supports <a href="#">SQL Server transactional replication</a> as a subscriber to replicate data. | Replication is not supported for Azure SQL Database Managed Instance. | Fully supports <a href="#">SQL Server transactional replication</a> , <a href="#">Always On Availability Groups</a> , Integration Services, and Log Shipping to replicate data. Also, traditional SQL Server backups are fully supported |
|  |  |   |  |

## Business motivations for choosing Azure SQL Database or SQL Server on Azure VMs

### 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 Server on Azure VMs. You also learn about calculating the total application cost.

### Billing and licensing basics

Currently, **SQL Database** is sold as a service and is available 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. With SQL Database 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](#). 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. For the latest information on the current supported service tiers, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#). You can also create [elastic pools](#) to share resources among database instances to reduce costs and accommodate usage spikes.

With **SQL Database**, 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 Server on Azure VMs**, 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.

### When using Azure SQL Database:

- Highly minimized administration costs

- Limited development costs for migrated applications
- SQL Database service costs
- No hardware purchasing costs

### **When using SQL Server on Azure VMs:**

- Higher administration costs
- Limited to no development costs for migrated applications
- Virtual Machine service costs
- SQL Server license costs
- No hardware purchasing 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 **Azure SQL Database**, you can continue to administer your database, but you no longer need to manage the database engine, server operating system or 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 Server on Azure VMs**, 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 Server running on Azure VMs**, 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 logical servers, elastic pools, and single databases** is 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 Database Managed Instance** greatly simplifies the migration of existing applications to Azure SQL Database, enabling you to bring migrated database applications to market in Azure quickly.

**SQL Server running on Azure VMs** is 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](#).

## 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.

# Create an Azure SQL database in the Azure portal

9/25/2018 • 4 minutes to read • [Edit Online](#)

This quickstart walks through how to create a SQL database in Azure using the [DTU-based purchasing model](#). Azure SQL Database is a “Database-as-a-Service” offering that enables you to run and scale highly available SQL Server databases in the cloud. This quickstart shows you how to get started by creating and then querying a SQL database using the Azure portal.

If you don't have an Azure subscription, create a [free account](#) before you begin.

## NOTE

This tutorial uses the DTU-based purchasing model but the [vCore-based purchasing model](#) is also available.

## Log in to the Azure portal

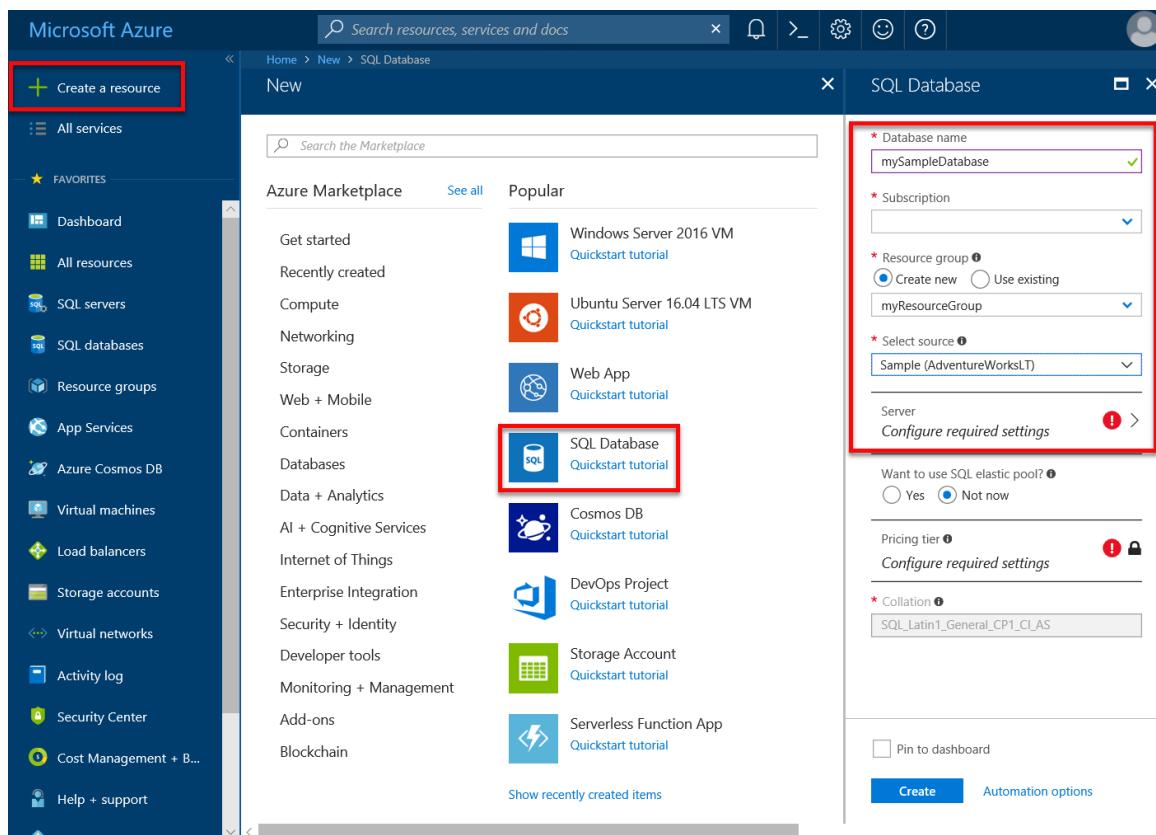
Sign in to the [Azure portal](#).

## Create a SQL database

An Azure SQL database is created with a defined set of [compute and storage resources](#). The database is created within an [Azure resource group](#) and in an [Azure SQL Database logical server](#).

Follow these steps to create a SQL database containing the Adventure Works LT sample data.

1. Click **Create a resource** in the upper left-hand corner of the Azure portal.
2. Select **Databases** from the **New** page, and select **Create** under **SQL Database** on the **New** page.



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>  | mySampleDatabase          | For valid database names, see <a href="#">Database Identifiers</a> .                |
| <b>Subscription</b>   | Your subscription         | For details about your subscriptions, see <a href="#">Subscriptions</a> .           |
| <b>Resource group</b> | myResourceGroup           | For valid resource group names, see <a href="#">Naming rules and restrictions</a> . |
| <b>Select source</b>  | Sample (AdventureWorksLT) | Loads the AdventureWorksLT schema and data into your new database                   |

**IMPORTANT**

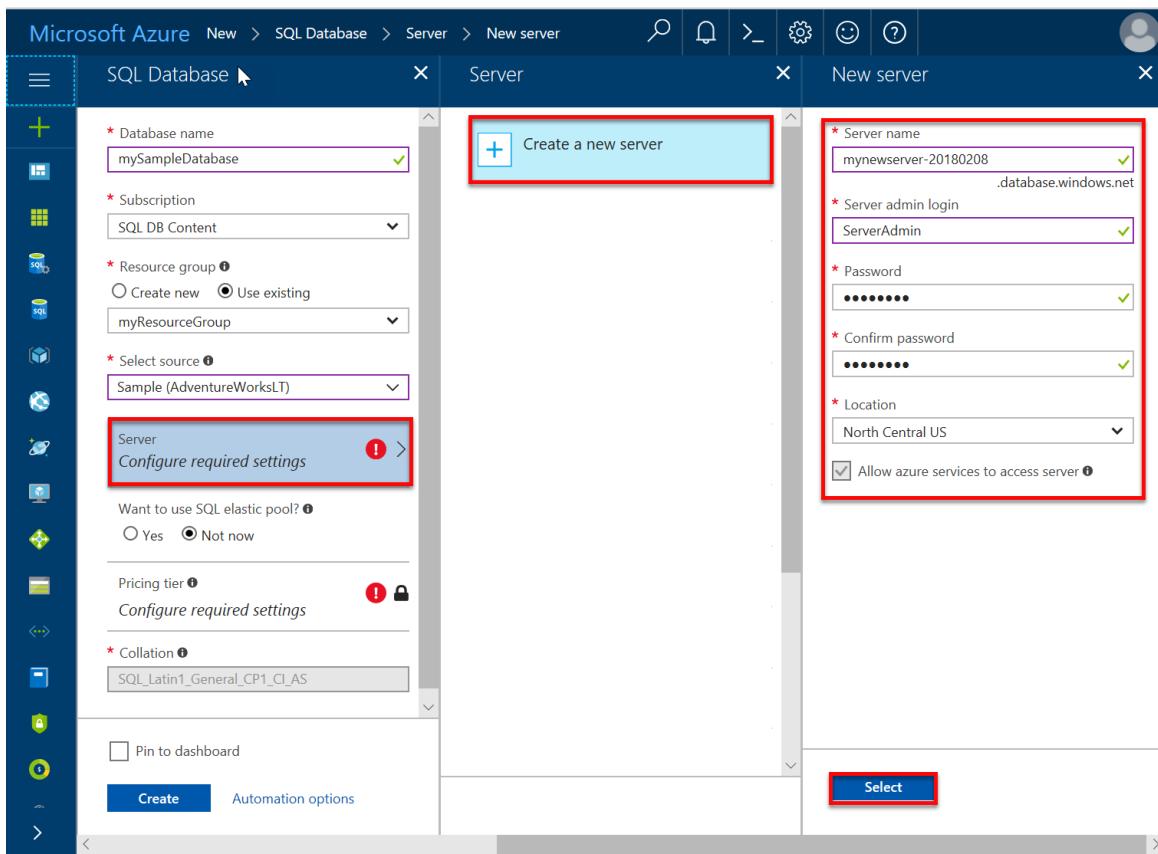
You must select the sample database on this form because it is used in the remainder of this quickstart.

4. Under **Server**, click **Configure required settings** and fill out the SQL server (logical server) form with the following information, as shown on the following image:

| 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 8 characters and must contain characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters. |
| <b>Subscription</b>       | Your subscription        | For details about your subscriptions, see <a href="#">Subscriptions</a> .   |
| <b>Resource group</b>     | myResourceGroup          | For valid resource group names, see <a href="#">Naming rules and restrictions</a> .   |
| <b>Location</b>           | Any valid location       | For information about regions, see <a href="#">Azure Regions</a> .  |

**IMPORTANT**

The server admin login and password that you specify here are required to log in to the server and its databases later in this quickstart. Remember or record this information for later use.



5. When you have completed the form, click **Select**.
6. Click **Pricing tier** to specify the service tier, the number of DTUs, and the amount of storage. Explore the options for the amount of DTUs and storage that is available to you for each service tier.

**IMPORTANT**

More than 1 TB of storage in the Premium tier is currently available in all regions except the following: UK North, West Central US, UK South2, China East, USDoDCentral, Germany Central, USDoDEast, US Gov Southwest, US Gov South Central, Germany Northeast, China North, US Gov East. In other regions, the storage max in the Premium tier is limited to 1 TB. See [P11-P15 Current Limitations](#).

7. For this quickstart, select the **Standard** service tier and then use the slider to select **10 DTUs (S0)** and **1 GB** of storage.

The screenshot shows the 'Configure performance' step in the Azure portal. The 'Standard' pricing tier is selected, highlighted with a red box. Below it, the DTU (10-3000 DTU) and Storage (100 MB-250 GB) sliders are also highlighted with red boxes. The monthly cost is listed as 15.00 USD.

- Accept the preview terms to use the **Add-on Storage** option.

#### IMPORTANT

More than 1 TB of storage in the Premium tier is currently available in all regions except the following: West Central US, China East, USDoDCentral, USGov Iowa, Germany Central, USDoDEast, US Gov Southwest, Germany Northeast, China North. In other regions, the storage max in the Premium tier is limited to 1 TB. See [P11-P15 Current Limitations](#).

- After selecting the server tier, the number of DTUs, and the amount of storage, click **Apply**.

- Now that you have completed the SQL Database form, click **Create** to provision the database.  
Provisioning takes a few minutes.

- On the toolbar, click **Notifications** to monitor the deployment process.

The screenshot shows the Microsoft Azure Dashboard. The Notifications panel is open, showing a single notification: 'Deployment in progress...' with a status of 'Running'. The notification text states: 'Deployment to resource group 'myResourceGroup' is in progress.'

## Query the SQL database

Now that you have created a sample database in Azure, let's use the built-in query tool within the Azure portal to confirm that you can connect to the database and query the data.

- On the SQL Database page for your database, click **Query editor (preview)** in the left-hand menu and

then click **Login**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'mySampleDatabase' node expanded, with 'Query editor (preview)' selected. The main area is titled 'mySampleDatabase - Query editor (preview)' and contains a 'Login' dialog. The 'Authorization type' dropdown is set to 'SQL server authentication'. The 'Login' field is populated with 'ServerAdmin' and the 'Password' field is filled with a masked password. Both fields are highlighted with a red border. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

2. Select SQL server authentication, provide the required login information, and then click **OK** to log in.
3. After you are authenticated as **ServerAdmin**, type 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. Click **Run** and then review the query results in the **Results** pane.

The screenshot shows the Microsoft Azure portal interface with the 'Query editor (preview)' page selected. The 'Run' button in the top right of the editor is highlighted with a red box. Below it, the 'Results' tab is selected. The results pane displays the output of the query:

| 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              |

At the bottom of the results pane, a green message bar says 'Query succeeded | 5s'.

5. Close the **Query editor** page, click **OK** to discard your unsaved edits.

## 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 wish to delete the resources that you created in this quickstart, use the following steps.

1. From the left-hand menu in the Azure portal, click **Resource groups** and then click **myResourceGroup**.
2. On your resource group page, click **Delete**, type **myResourceGroup** in the text box, and then click **Delete**.

## Next steps

- Now that you have a database, you need to create a server-level firewall rule to connect to it from your on-premises tools. See [Create server-level firewall rule](#)
- If creating a server-level firewall rule, 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 create databases using Azure CLI, see [Azure CLI samples](#)
- To create databases using Azure PowerShell, see [Azure PowerShell samples](#)

# Create an Azure SQL Database Managed Instance

10/26/2018 • 3 minutes to read • [Edit Online](#)

This quickstart walks through how to create an Azure SQL Database [Managed Instance](#) in the Azure portal.

If you don't have an Azure subscription, create a [free](#) account before you begin.

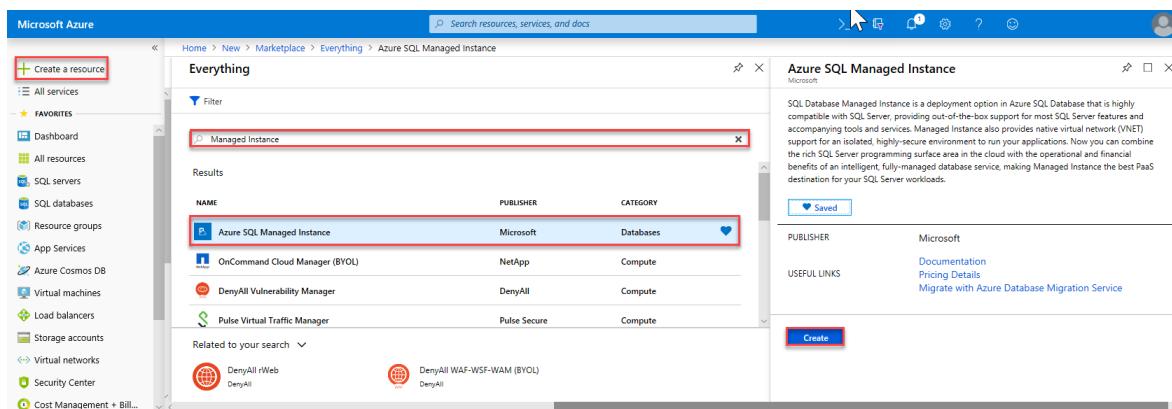
## Sign in to the Azure portal

Sign in to the [Azure portal](#).

## Create a Managed Instance

The following steps show you how to create a Managed Instance.

1. Click **Create a resource** in the upper left-hand corner of the Azure portal.
2. Locate **Managed Instance** and then select **Azure SQL Managed Instance**.
3. Click **Create**.



4. Fill out the Managed Instance form with the requested information, using the information in the following table:

| SETTING                             | SUGGESTED VALUE     | DESCRIPTION  |
|-------------------------------------|---------------------|--|
| <b>Subscription</b>                 | Your subscription   | A subscription in which you have permission to create new resources  |
| <b>Managed instance name</b>        | Any valid name      | For valid names, see <a href="#">Naming rules and restrictions</a> .   |
| <b>Managed instance admin login</b> | Any valid user name | For valid names, see <a href="#">Naming rules and restrictions</a> . Do not use "serveradmin" as that is 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> .                        |

| SETTING                | SUGGESTED VALUE  | DESCRIPTION  |
|------------------------|--|--|
| <b>Resource Group</b>  | A new or existing resource group   | For valid resource group names, see <a href="#">Naming rules and restrictions</a> .  |
| <b>Location</b>        | The location in which you want to create the Managed Instance  | For information about regions, see <a href="#">Azure Regions</a> .   |
| <b>Virtual network</b> | Select either <b>Create new virtual network</b> or a virtual network that you previously created in the resource group that you previously provided in this form | To configure a virtual network for a Managed Instance with custom settings, see <a href="#">Configure SQL Managed Instance virtual network environment template</a> in Github. For information regarding the requirements for configuring the network environment for a Managed Instance, see <a href="#">Configure a VNet for Azure SQL Database Managed Instance</a> |

The screenshot shows the 'Configure performance' step of the Azure SQL Managed Instance creation wizard. Key visible elements include:

- Compute Generation:** Gen4 is selected.
- vCores:** A slider is set to 16.
- Storage:** A slider is set to 32 GB.
- Days of backup retention:** Free.
- Pricing tier:** General Purpose: 32 GB, 16 vCores.
- Buttons:** 'Create' (highlighted with a red box) and 'Automation options'.

- Click **Pricing tier** to size compute and storage resources as well as review the pricing tier options. The General Purpose pricing tier with 32 GB of memory and 16 vCores is the default value.
- Use the sliders or text boxes to specify the amount of storage and the number of virtual cores.
- When complete, click **Apply** to save your selection.
- Click **Create** to deploy the Managed Instance.
- Click the **Notifications** icon to view the status of deployment.



10. Click **Deployment in progress** to open the Managed Instance window to further monitor the deployment progress.

#### IMPORTANT

For the first instance in a subnet, deployment time is typically much longer than in case of the subsequent instances. Do not cancel deployment operation because it lasts longer than you expected. Creating the second Managed Instance in the subnet only takes a couple of minutes.

## Review resources and retrieve your fully-qualified server name

After the deployment completes successfully, review the resources created and retrieve the fully qualified server name for use in later quickstarts.

1. Open the resource group for your Managed Instance and view its resources that were created for you in the [Create a Managed Instance](#) quickstart.

A screenshot of the Microsoft Azure Resource Groups page. The URL is "Home > Resource groups > mymirmg". On the left, a sidebar lists options like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Resource costs, Deployments, Policies, Properties, Locks, and Automation script. The "Overview" option is highlighted with a red box. The main pane shows the "Subscription (change)" and "Subscription ID". Under "Deployments", it says "1 Succeeded". Under "Tags", there's a link "Click here to add tags". Below that is a table of resources:

| NAME                          | TYPE                 |
|-------------------------------|----------------------|
| mymanagedinstance             | SQL managed instance |
| rt-mymanagedinstance          | Route table          |
| VirtualClusterManagedInstance | Virtual cluster      |
| vnet-mymanagedinstance        | Virtual network      |

The "mymanagedinstance" row has a checked checkbox next to it and is highlighted with a red box.

Open your Managed Instance resource in the Azure portal.

2. Click your Managed Instance.
3. On the **Overview** tab, locate the **Host** property and copy the fully-qualified host address for the Managed Instance.

The screenshot shows the Azure portal interface for managing a SQL Managed Instance named 'mymanagedinstance'. The 'Essentials' tab is active, displaying basic information such as Resource group, Status, Location, and Subscription name. A tooltip for the 'Host' field shows the full connection string: 'mymanagedinstance .neu13937e49e10d2.database.windows.net'. This string is also highlighted with a red box.

The name is similar to this: **your\_machine\_name.neu15011648751ff.database.windows.net**.

## Next steps

- To learn about connecting to a Managed Instance, see:
  - For an overview of the connection options for applications, see [Connect your applications to Managed Instance](#).
  - For a quickstart showing how to connect to a Managed Instance from an Azure virtual machine, see [Configure an Azure virtual machine connection](#).
  - For a quickstart showing how to connect to a Managed Instance from an on-premises client computer 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, you can use the [Azure Database Migration Service \(DMS\) for migration](#) to restore from a database backup file or 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 using Azure SQL Analytics](#)

# Create a server-level firewall rule for your SQL database using the Azure portal

9/25/2018 • 2 minutes to read • [Edit Online](#)

This quickstart walks through how to create a server-level firewall rule for an Azure SQL database to enable you to connect to it from an on-premises resource.

## Prerequisites

This quickstart uses as its starting point the resources created in this quickstart: [Create an Azure SQL database in the Azure portal](#)

## Log in to the Azure portal

Sign in to the [Azure portal](#).

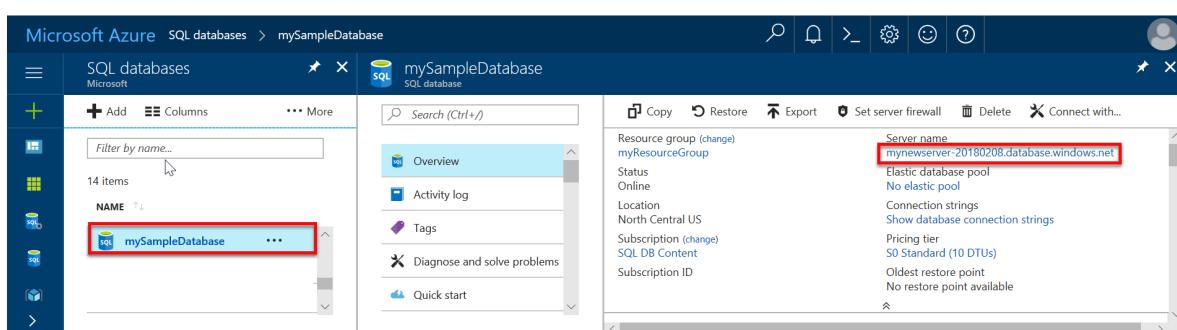
## Create a server-level firewall rule

The SQL Database service creates a firewall at the server-level that prevents applications and tools from connecting to the server or any databases on the server unless a firewall rule is created to open the firewall. For a connection from an IP address outside of Azure, create a firewall rule for a specific IP address or range of addresses. Follow these steps to create a [SQL Database server-level firewall rule](#) for your client's IP address and enable external connectivity through the SQL Database firewall for your IP address only.

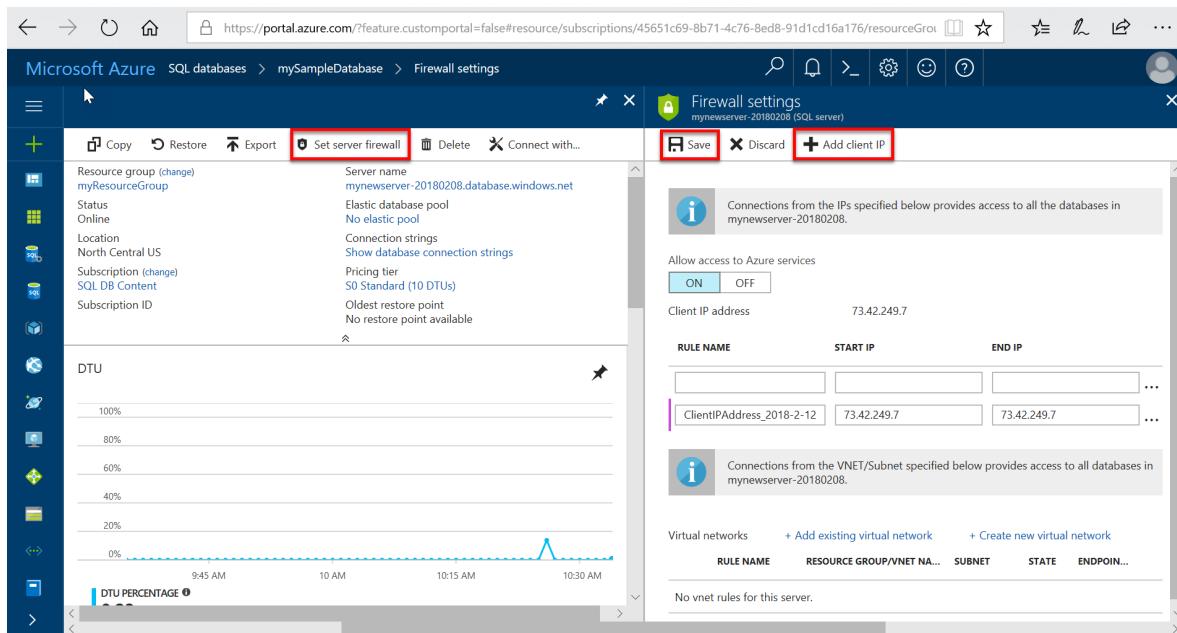
### NOTE

SQL Database communicates over port 1433. If you are 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 cannot connect to your Azure SQL Database server unless your IT department opens port 1433.

1. After the deployment completes, click **SQL databases** from the left-hand menu and then click **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 for use to connect to your server and its databases in subsequent quickstarts.



3. Click **Set server firewall** on the toolbar as shown in the previous image. 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 firewall rule. A firewall rule can open port 1433 for a single IP address or a range of IP addresses.
5. Click **Save**. A server-level firewall rule is created for your current IP address opening port 1433 on the logical server.
6. Click **OK** and then close the **Firewall settings** page.

You can now connect to the SQL Database server and its databases using SQL Server Management Studio or another tool of your choice from this IP address using the server admin account created previously.

#### IMPORTANT

By default, access through the SQL Database firewall is enabled for all Azure services. Click **OFF** on this page to disable for all Azure services.

## 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 wish to delete the resources that you created in this quickstart, use the following steps.

1. From the left-hand menu in the Azure portal, click **Resource groups** and then click **myResourceGroup**.
2. On your resource group page, click **Delete**, type **myResourceGroup** in the text box, and then click **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 Azure SQL database using SSMS](#)
  - [Design an Azure SQL database and connect with C# and ADO.NET](#)

# Configure Azure VM to connect to an Azure SQL Database Managed Instance

9/25/2018 • 5 minutes to read • [Edit Online](#)

This quickstart demonstrates 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 as its starting point the resources created in this quickstart: [Create a Managed Instance](#).

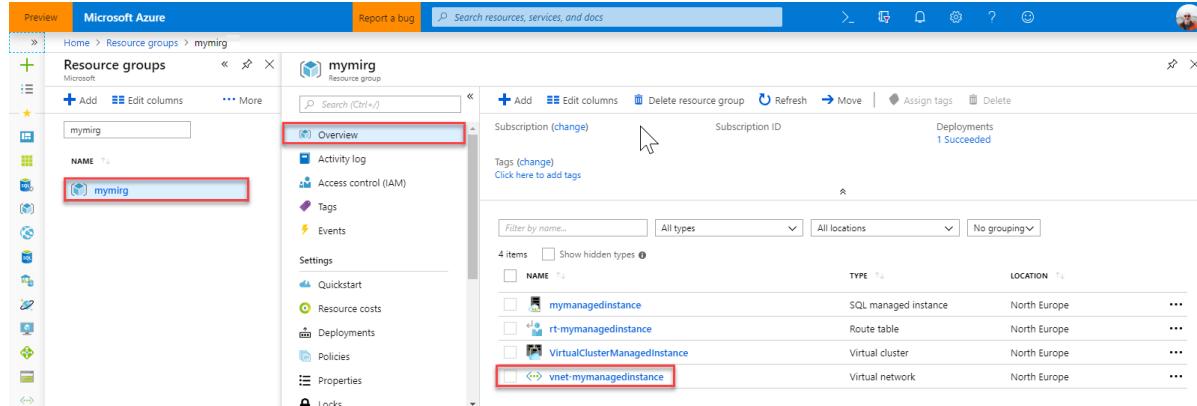
## Sign in to the Azure portal

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 for an Azure virtual machine to connect to the Managed Instance. The Managed Instance subnet is dedicated to Managed Instances and you cannot create any other resources (for example 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 and click the virtual network for your Managed Instance and then click **Subnets**.



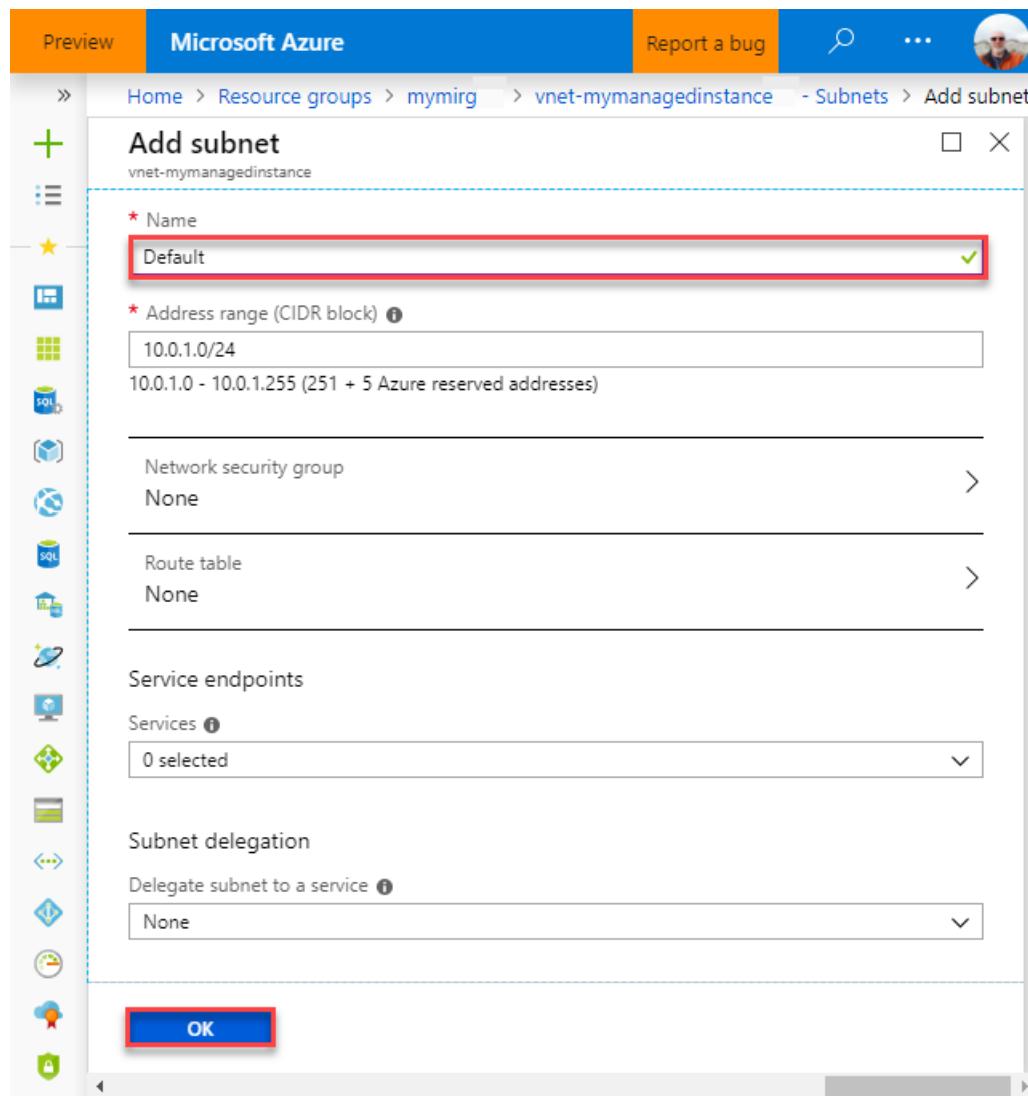
| NAME                          | TYPE                 | LOCATION     |
|-------------------------------|----------------------|--------------|
| mymanagedinstance             | SQL managed instance | North Europe |
| rt-mymanagedinstance          | Route table          | North Europe |
| VirtualClusterManagedInstance | Virtual cluster      | North Europe |
| vnet-mymanagedinstance        | Virtual network      | North Europe |

2. Click the + sign next to **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 'Home > Resource groups > mymigr > vnet-mymanagedinstance - Subnets'. On the left, a sidebar lists options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Address space, Connected devices, and Subnets. The 'Subnets' option is selected and highlighted with a red box. The main content area displays a table titled 'Subnets' with one row: 'ManagedInstance' with address range '10.0.0.0/24' and available addresses '251'. There are also 'Search subnets' and 'Create' buttons.

- Fill out the form with the requested information, using the information in the following 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.                       |



4. Click **OK** to create this additional subnet in the Managed Instance VNet.

## Create a virtual machine in the new subnet in the VNet

The following steps show you how to create a virtual machine in the new subnet to connect to the Managed Instance.

### Prepare the Azure virtual machine

Since SQL Managed Instance is placed in your private Virtual Network, you need to create an Azure VM with some installed SQL client tool like SQL Server Management Studio or Azure Data Studio to connect to the Managed Instance and execute queries. This quickstart uses SQL Server Management Studio.

The easiest way to create a client virtual machine with all necessary tools is to use the Azure Resource Manager templates.

1. Click on the following button to create a client virtual machine and install SQL Server Management Studio (make sure that you are signed-in to the Azure portal in another browser tab):



2. Fill out the form with the requested information, using the information in the following table:

| 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 must be the resource group 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 user name  | For valid names, see <a href="#">Naming rules and restrictions</a> . Do not use "serveradmin" as that is a reserved server-level role. |
| <b>Password</b>                     | Any valid password   | The password must be at least 12 characters long and meet the <a href="#">defined complexity requirements</a> .                        |
| <b>Virtual Machine Size</b>         | Any valid size   | The default in this template of **Standard_B2s is sufficient for this quickstart.  |
| <b>Location</b>                     | [resourceGroup0.location].   | Do not change this value   |
| <b>Virtual Network Name</b>         | The location that you previously selected  | For information about regions, see <a href="#">Azure Regions</a> .   |
| <b>Subnet name</b>                  | The name of the subnet that you created in the previous procedure                                | Do not choose the subnet in which you created the Managed Instance   |
| <b>artifacts Location</b>           | [deployment().properties.templateLink.uri] Do not change this value                              |  |
| <b>artifacts Location Sas token</b> | leave blank  | Do not change this value   |

The screenshot shows the Microsoft Azure portal interface for creating a custom deployment. The top navigation bar includes 'Preview' (highlighted in orange), 'Microsoft Azure', 'Report a bug', and various icons for search, refresh, and notifications.

**TEMPLATE**

- 201-vm-win-vnet-sql-tools (4 resources)

Actions: Edit template, Edit parameters, Learn more

**BASICS**

- \* Subscription: mymirmg
- \* Resource group: mymirmg
- \* Location: North Europe

**SETTINGS**

- \* Virtual Machine Name: mymimvms
- \* Admin Username: mymimvmsadmin
- \* 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 terms and conditions stated above** checkbox.
4. Click **Purchase** to deploy the Azure VM in your network.
5. Click the **Notifications** icon to view the status of deployment.

Do not continue until the Azure virtual machine is created.

## 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. At the top, there's a navigation bar with icons for back, forward, home, and search, followed by the URL <https://ms.portal.azure.com/#@microsoft.onmicrosoft.com/resource/subscriptions/45651c69-8b7>. Below the navigation is a blue header bar with the text "Microsoft Azure" and a "Report a bug" link. To the right of the header are various icons for reporting bugs, sharing, and more.

The main content area shows a list of resources under the "mymirg" resource group. The first item is "mymivmq" (Virtual machine), which is highlighted with a red box. Below it is a search bar with the placeholder "Search (Ctrl+/" and a "Connect" button also highlighted with a red box. To the right of the search bar are buttons for "Start", "Restart", "Stop", "Capture", "Delete", and "Refresh".

The left sidebar contains a tree view of the resource group structure, with "mymivmq" expanded to show its sub-components: "Overview", "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", and "Automation script".

The "Overview" section displays detailed information about the virtual machine, including:

- Resource group: mymirg
- Status: Running
- Location: North Europe
- Subscription: (change)
- Subscription ID
- Tags: (change) Click here to add tags
- Computer name: mymivmq
- Operating system: Windows
- Size: Standard B2s (2 vcpus, 4 GB memory)
- Public IP address
- Virtual network/subnet: vnet-mymanagedinstance /Default
- DNS name: Configure

At the bottom of the overview section, there are buttons for "1 hour", "6 hours", "12 hours", "1 day", "7 days", and "30 days" to filter performance data. Below this are two line charts: "CPU (average)" and "Network (total)".

2. Click **Connect**.

A Remote Desktop Protocol file (.rdp file) form appears with the public IP address and port number for the virtual machine.

The dialog box has a title "Connect to virtual machine" and a close button "X". It includes tabs for "RDP" (selected) and "SSH".

The "RDP" tab contains fields for "IP address" (set to "Public IP address (40.115.106.63)") and "Port number" (set to "3389").

Below these fields is a button "Download RDP File" highlighted with a red box.

At the bottom, there's a note: "Inbound traffic to the Public IP address may be blocked. You can update inbound port rules in the [VM Networking](#) page." This note has an info icon.

At the very bottom, there's another note: "You can troubleshoot VM connection issues by opening the [Diagnose and solve problems](#) page." This note has a wrench icon.

3. Click **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, click **Connect**. On a Mac, you need an RDP client such as this [Remote Desktop Client](#) from the Mac App Store.
7. Enter the user name and password you specified when creating the virtual machine, then click **Ok**.
8. You may receive a certificate warning during the sign-in process. Click **Yes** or **Continue** to proceed with the connection.

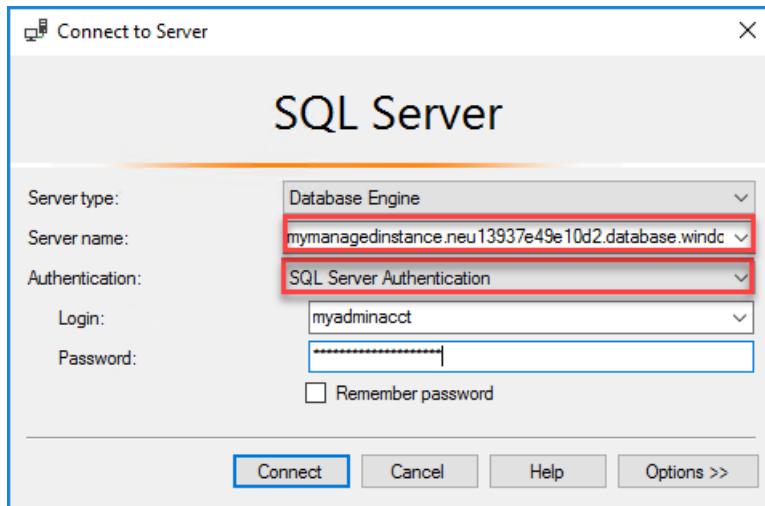
You are connected to your virtual machine in the Server Manager dashboard.

## Use SSMS to connect to the Managed Instance

1. In the virtual machine, open SQL Server Management Studio (SSMS).

It will take a few moments to open as it needs to complete its configuration as 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 login and password, and then click **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](#) to restore from a database backup file or the [T-SQL RESTORE command](#) to restore from a database backup file.

# Configure a point-to-site connection to an Azure SQL Database Managed Instance from on-premises

10/8/2018 • 2 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 as its starting point the resources created in this quickstart: [Create a Managed Instance](#).
- Requires PowerShell 5.1 and Azure PowerShell 5.4.2 or higher your on-premises client computer.
- 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 and paste 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 performs the following three steps:
  - 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>'
}

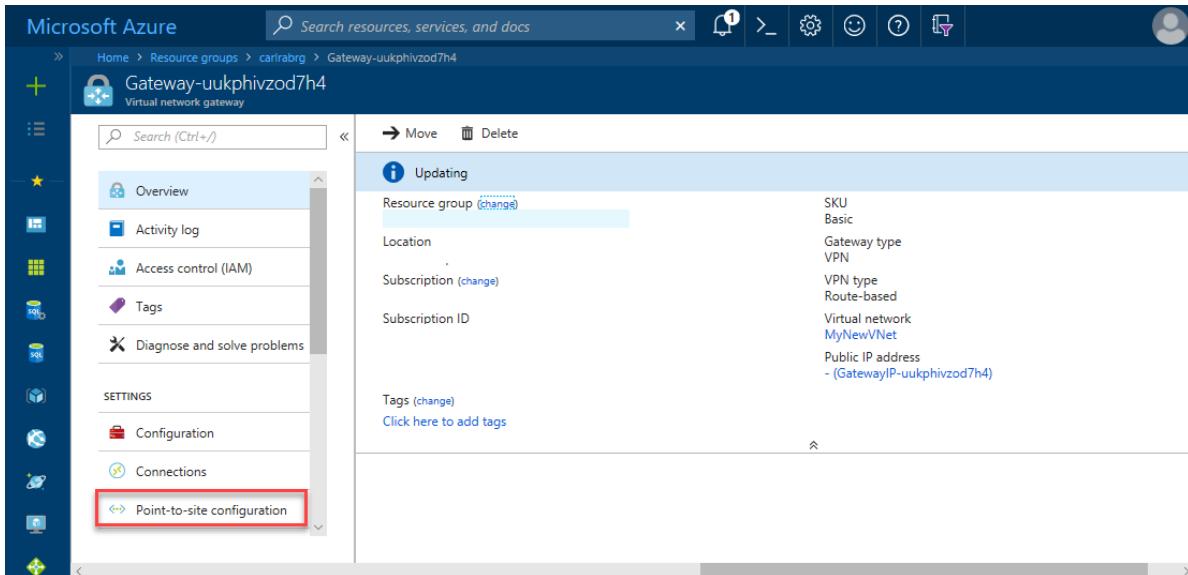
Invoke-Command -ScriptBlock ([Scriptblock]::Create((iwr ($scriptUrlBase+'/attachVPNGateway.ps1?`t=' + [DateTime]::Now.Ticks)).Content)) -ArgumentList $parameters, $scriptUrlBase
```

3. Provide the requested parameters in the PowerShell script. The values for `<subscriptionId>`, `<resourceGroup>` and `<virtualNetworkName>` should match the ones that are used in [Create Managed Instance](#) quickstart. The value for `<certificateNamePrefix>` can be a string of your choice.
4. Execute the PowerShell script.

## Create a VPN connection to your Managed Instance

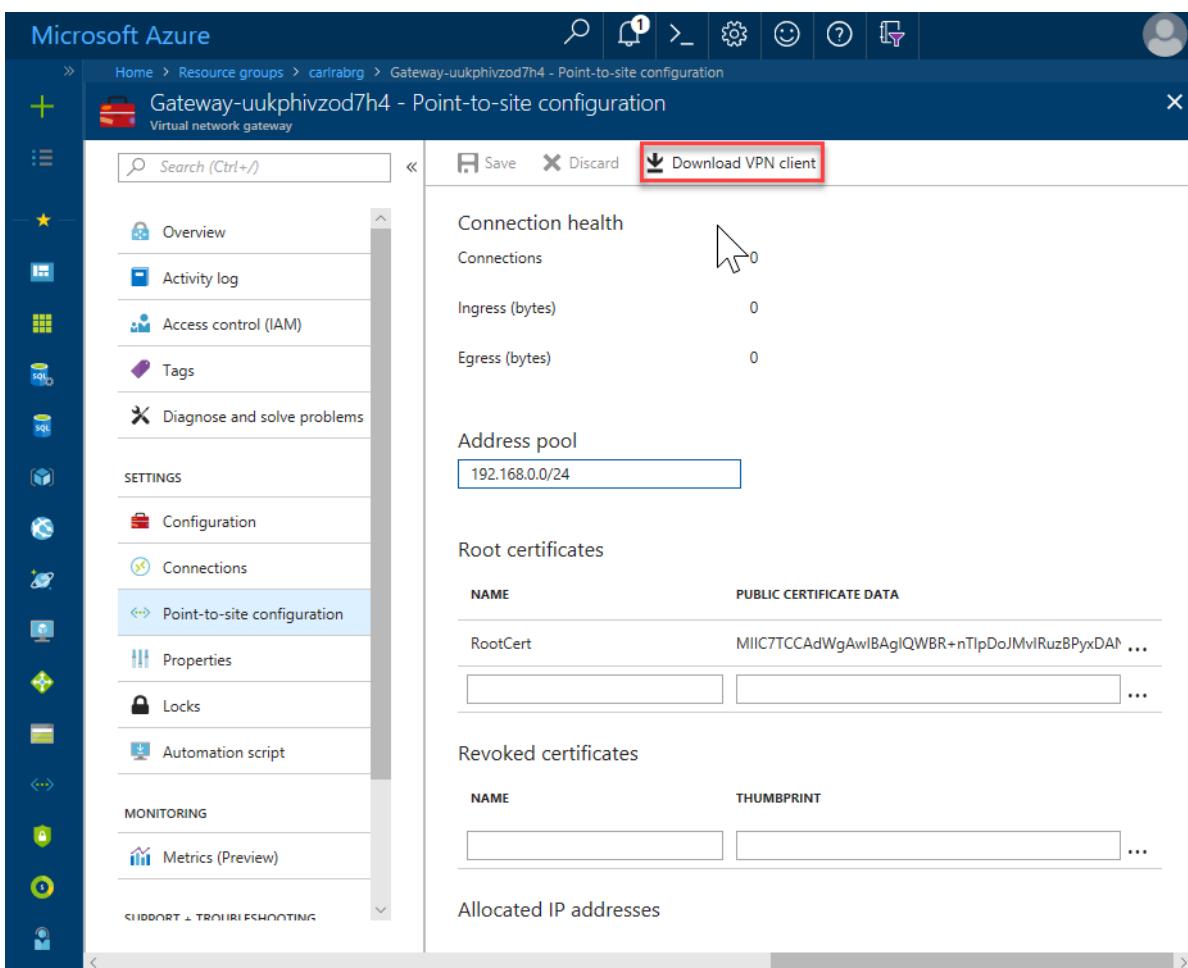
1. Sign in to the [Azure portal](#).
2. Open the resource group in which you created the virtual network gateway and then open the virtual

network gateway resource.



The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes 'Search resources, services, and docs', a user profile icon, and various navigation icons. The main title is 'Gateway-uukphivzod7h4' under 'Virtual network gateway'. On the left, a sidebar lists options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration, Connections, and Point-to-site configuration (which is highlighted with a red box). The main content area shows details for the gateway, including Resource group (carlrbrg), Location (United Kingdom), Subscription (MyNewVNet), SKU (Basic), Gateway type (VPN), VPN type (Route-based), and Virtual network (MyNewVNet). It also shows a Public IP address and a note about tags.

3. Click **Point-to-site configuration** and then click **Download VPN client**.



This screenshot shows the 'Point-to-site configuration' page for the same gateway. The left sidebar includes Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration, Connections, Point-to-site configuration (highlighted with a red box), Properties, Locks, and Automation script. The main content area displays Connection health (0 connections), Address pool (192.168.0.0/24), Root certificates (listing RootCert with certificate data), Revoked certificates, and Allocated IP addresses. A prominent 'Download VPN client' button is highlighted with a red box at the top right of the content area.

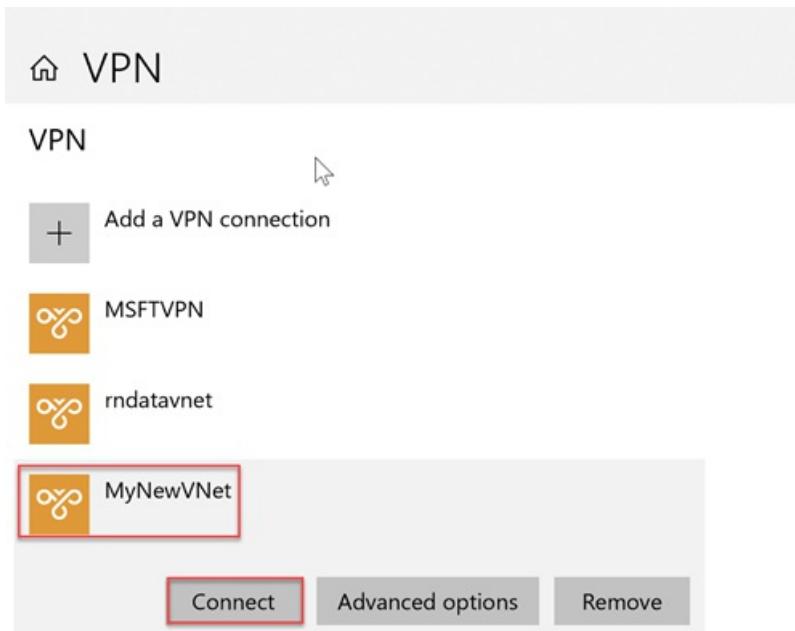
4. Extract the files from the zip file and then open the extracted folder.
5. Navigate to the WindowsAmd64 folder and open the **VpnClientSetupAmd64.exe** file.
6. If you receive a **Windows protected your PC** message, click **More info** and then click **Run anyway**.



7. Click **Yes** in the User Account Control dialog box to proceed.
8. In the MyNewVNet dialog box, click **Yes** to install a Vpn Client for MyNewVNet.

## Connect to the VPN connection

1. Go to VPN connections on your client computer and click **MyNewVNet** to establish a connection to this VNet.

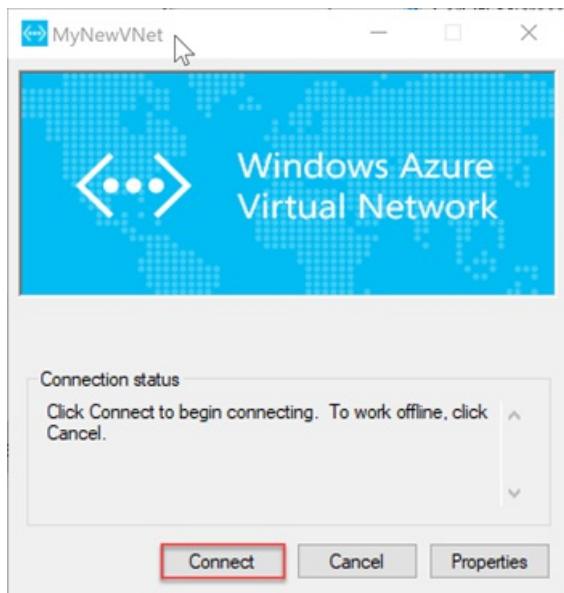


### Advanced Options

Allow VPN over metered networks

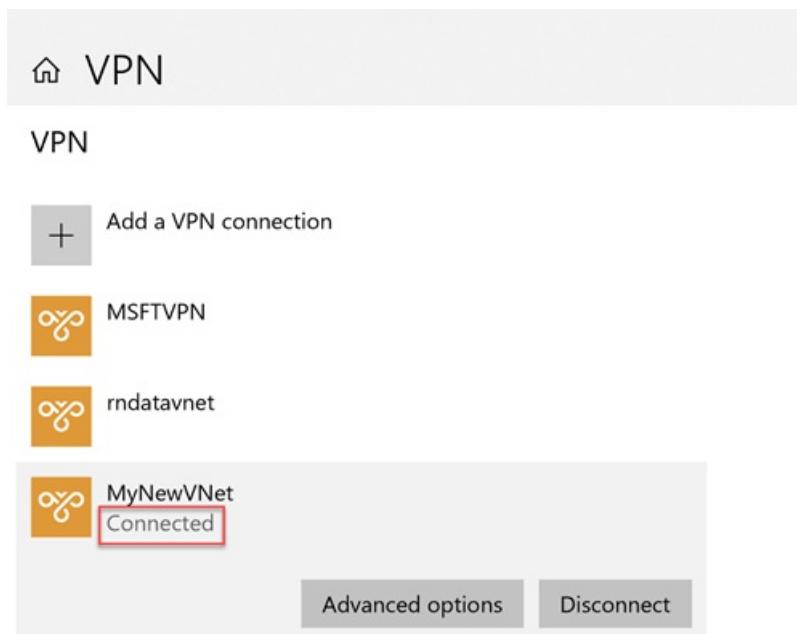
2. Click **Connect**.

3. In the MyNewVNet dialog box, click **Connect**.



4. When prompted that Connection Manager needs elevated privilege to update your route table, click **Continue**.

5. Click **Yes** in the User Account Control dialog box to proceed.



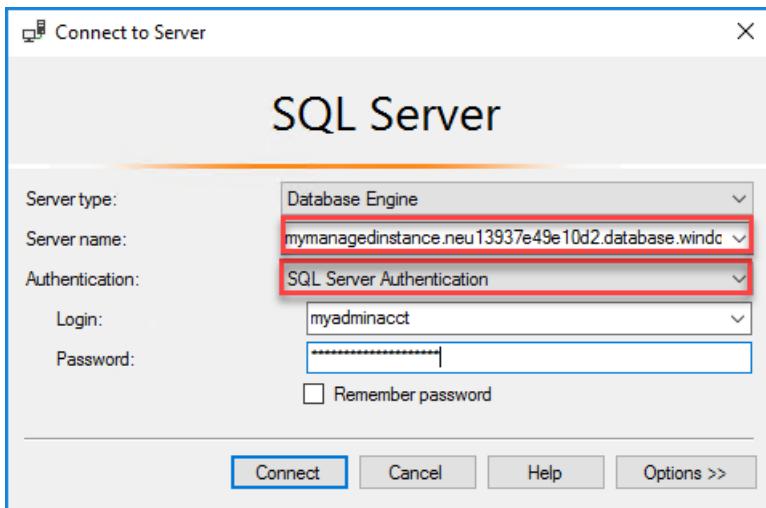
## Advanced Options

Allow VPN over metered networks

You have 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, select **SQL Server Authentication**, provide your login and password, and then click **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 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](#) to restore from a database backup file or the [T-SQL RESTORE command](#) to restore from a database backup file.

# Import a BACPAC file to a new Azure SQL Database

10/19/2018 • 4 minutes to read • [Edit Online](#)

When you need to import a database from an archive or when migrating from another platform, you can import the database schema and data from 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 imported from Azure blob storage (standard storage only) or from local storage in an on-premises location. To maximize the import speed, we recommend that you specify a higher service tier and compute size, such as a P6, and then scale to down as appropriate after the import is successful. Also, the database compatibility level after the import is based on the compatibility level of the source database.

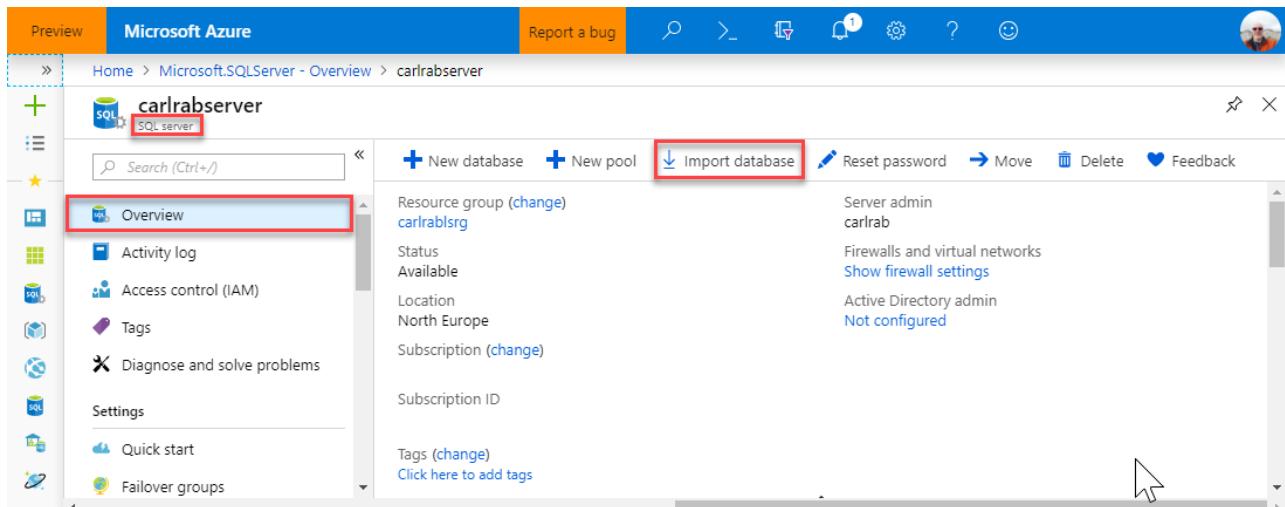
## IMPORTANT

After you migrate your database to Azure SQL 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.

## Import from a BACPAC file using Azure portal

This article provides directions for creating an Azure SQL database from a BACPAC file stored in Azure blob storage using the [Azure portal](#). Import using the Azure portal only supports importing a BACPAC file from Azure blob storage.

To import a database using the Azure portal, open the page for the server (not the page for the database) to associate the database to and then click **Import** on the toolbar. Specify the storage account and container and select the BACPAC file you want to import. Select the size of the new database (usually the same as origin) and provide the destination SQL Server credentials.



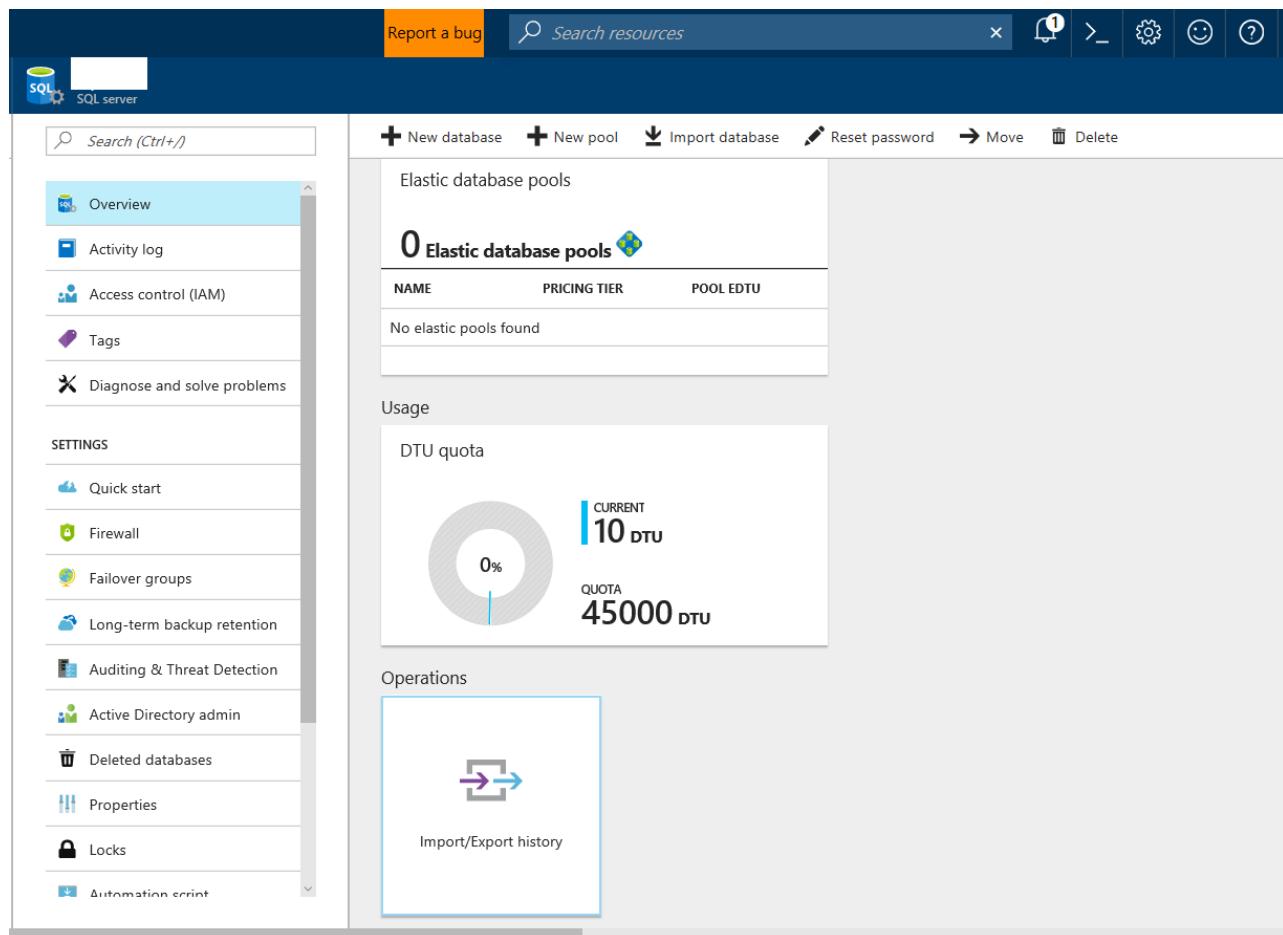
To monitor the progress of the import operation, open the page for the logical server containing the database being imported. Scroll down to **Operations** and then click **Import/Export history**.

## NOTE

Azure SQL Database Managed Instance supported importing from a BACPAC file using the other methods in this article but does not currently support migrating using the Azure portal.

## Monitor the progress of an import operation

To monitor the progress of the import operation, open the page for the logical server into which the database is being imported. Scroll down to **Operations** and then click **Import/Export** history.



The screenshot shows the Azure SQL Server management portal. The left sidebar contains navigation links such as Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Quick start, Firewall, Failover groups, Long-term backup retention, Auditing & Threat Detection, Active Directory admin, Deleted databases, Properties, Locks, Automation script), and a search bar. The main content area has a header with buttons for New database, New pool, Import database, Reset password, Move, and Delete. Below this is a section titled "Elastic database pools" showing 0 elastic database pools. Further down is a "Usage" section with a DTU quota meter showing 0% usage, current 10 DTU, and a quota of 45000 DTU. At the bottom is an "Operations" section with a link to "Import/Export history".

| Import/Export history |                          |           |                      | Properties |
|-----------------------|--------------------------|-----------|----------------------|------------|
|                       |                          |           |                      |            |
| Import/Export history |                          |           |                      |            |
| OPERATION             | DATABASE                 | STATUS    | COMPLETION TIME      |            |
| Import                | carlpaasdb-restore fr... | Completed | 4/11/2016 3:21:52 PM |            |
| Export                | carlpaasdb               | Completed | 4/6/2016 3:54:00 PM  |            |

To verify the database is live on the server, click **SQL databases** and verify the new database is **Online**.

## Import from a BACPAC file using SQLPackage

To import a SQL database using the [SqlPackage](#) command-line utility, see [Import 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](#).

See the following SQLPackage command for a script example for how to import the **AdventureWorks2008R2** database from local storage to an Azure SQL Database logical server, called **mynewserver20170403** in this example. This script shows the creation of a new database called **myMigratedDatabase**, with a service tier of **Premium**, and a Service Objective of **P6**. Change these values as appropriate to your environment.

```
SqlPackage.exe /a:import /tcs:"Data Source=mynewserver20170403.database.windows.net;Initial Catalog=myMigratedDatabase;User Id=ServerAdmin;Password=<change_to_your_password>" /sf:AdventureWorks2008R2.bacpac /p:DatabaseEdition=Premium /p:DatabaseServiceObjective=P6
```

### IMPORTANT

An Azure SQL Database logical server listens on port 1433. If you are attempting to connect to an Azure SQL Database logical server from within a corporate firewall, this port must be open in the corporate firewall for you to successfully connect.

This example shows how to import a database using SqlPackage.exe 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"
```

## Import from a BACPAC file using PowerShell

Use the [New-AzureRmSqlDatabaseImport](#) cmdlet to submit an import database request to the Azure SQL Database service. Depending on the size of your database, the import operation may take some time to complete.

```
$importRequest = New-AzureRmSqlDatabaseImport -ResourceGroupName "myResourceGroup" `  
    -ServerName $servername `  
    -DatabaseName "MyImportSample" `  
    -DatabaseMaxSizeBytes "262144000" `  
    -StorageKeyType "StorageAccessKey" `  
    -StorageKey $(Get-AzureRmStorageAccountKey -ResourceGroupName "myResourceGroup" -StorageAccountName  
$storageaccountname).Value[0] `  
    -StorageUri "http://$storageaccountname.blob.core.windows.net/importsample/sample.bacpac" `  
    -Edition "Standard" `  
    -ServiceObjectiveName "P6" `  
    -AdministratorLogin "ServerAdmin" `  
    -AdministratorLoginPassword $(ConvertTo-SecureString -String "ASecureP@ssw0rd" -AsPlainText -Force)
```

To check the status of the import request, use the [Get-AzureRmSqlDatabaseImportExportStatus](#) cmdlet. Running this immediately after the request usually returns **Status: InProgress**. When you see **Status: Succeeded** the import is complete.

```
$importStatus = Get-AzureRmSqlDatabaseImportExportStatus -OperationStatusLink  
$importRequest.OperationStatusLink  
[Console]::Write("Importing")  
while ($importStatus.Status -eq "InProgress")  
{  
    $importStatus = Get-AzureRmSqlDatabaseImportExportStatus -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

- Import to a database in elastic pool is not supported. You can import data into a single database and then move the database to a pool.

## Import using other methods

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 [Connect to SQL Database with SQL Server Management Studio and perform a sample T-SQL query](#).
- 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 [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](#).

# Restore a database backup to an Azure SQL Database Managed Instance

10/16/2018 • 2 minutes to read • [Edit Online](#)

This quickstart demonstrates how to restore a backup of a database stored in Azure blob storage into the Managed Instance using the Wide World Importers - Standard backup file. This method requires some downtime.

For a tutorial using the Azure Database Migration Service (DMS) for migration, see [Managed Instance migration using DMS](#). For a discussion of the various migration methods, see [SQL Server instance migration to Azure SQL Database Managed Instance](#).

## Prerequisites

This quickstart:

- Uses as its starting point the resources created in this quickstart: [Create a Managed Instance](#).
- Requires the newest version of [SQL Server Management Studio](#) on your on-premises client computer
- Requires connectivity to your Managed Instance using SQL Server Management Studio. See these quickstarts for connectivity options:
  - [Connect to an Azure SQL Database Managed Instance from an Azure VM](#)
  - [Connect to an Azure SQL Database Managed Instance from on-premises using a Point-to-Site connection](#).
- Uses a preconfigured Azure blob storage account containing the Wide World Importers - Standard backup file (downloaded from <https://github.com/Microsoft/sql-server-samples/releases/download/wide-world-importers-v1.0/WideWorldImporters-Standard.bak>).

### NOTE

For more information about backing up and restoring a SQL Server database using Azure blob storage and a Shared Access Signature (SAS), see [SQL Server Backup to URL](#).

## Restore the Wide World Importers database from a backup file

With SSMS, use the following steps to restore the Wide World Importers database to your Managed Instance from the backup file.

1. Open SQL Server Management Studio (SSMS) and connect to your Managed Instance.
2. In SSMS, open a new query window.
3. Use the following script to create a credential in the Managed Instance using the preconfigured storage account and SAS key.

```
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'
```

```
CREATE CREDENTIAL [https://mitutorials.blob.core.windows.net/databases]
    WITH IDENTITY = 'SHARED ACCESS SIGNATURE'
    , SECRET = 'sv=2017-11-09&ss=bfqt&srt=sco&sp=rw&dlacup&se=2028-09-06T02:52:55Z&st=2018-09-04T18:52:55Z&s='
```

100 %

Messages

Commands completed successfully.

100 %

Query executed successfully. | master | 00:00:00 | 0 rows

#### NOTE

Always remove the leading ? from generated SAS key.

4. Use the following script to check the SAS credential and backup validity - providing the URL for the container with the backup file:

```
RESTORE FILELISTONLY FROM URL =
    'https://mitutorials.blob.core.windows.net/databases/WideWorldImporters-Standard.bak'
```

|   | LogicalName  | PhysicalName                            | Type | FileGroupName | Size       | MaxSize        | FileId | CreateLSN         | DropLSN | Uniquid         |
|---|--------------|---|------|---------------|------------|----------------|--------|-------------------|---------|-----------------|
| 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       | 28D406E0-78F4-4 |
| 3 | WWI_Log      | E:\Log\WideWorldImporters.ldf           | L    | NULL          | 104857600  | 2199023255552  | 2      | 0                 | 0       | 6AC6807E-8774-4 |

100 %

Results

Query executed successfully. | master | 00:00:03 | 3 rows

5. Use the following script to restore the Wide World Importers database from a backup file - providing the URL for the container with the backup file:

```
RESTORE DATABASE [Wide World Importers] FROM URL =
    'https://mitutorials.blob.core.windows.net/databases/WideWorldImporters-Standard.bak'
```

100 %

Messages

Commands completed successfully.

100 %

Query executed successfully. | master | 00:01:45 | 0 rows

6. To track the status of your restore, run the following query in a new query session:

```
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')`
```

- When the restore completes, view it in Object Explorer.

## Next steps

- For troubleshooting backup to URL, see [SQL Server Backup to URL Best Practices and Troubleshooting](#).
- For an overview of the connection options for applications, see [Connect your applications to Managed Instance](#).
- To query using one of your favorite tools or languages, see [connect and query](#).

# Azure SQL Database Connect and Query Quickstarts

9/24/2018 • 2 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

| <a href="#">SQL Server Management Studio</a> | 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.                         |
|--|---|
| <a href="#">Azure Data Studio</a>            | 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. |
| <a href="#">Azure portal</a>                 | 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.                    |
| <a href="#">Visual Studio Code</a>           | 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.           |
| <a href="#">.NET with Visual Studio</a>      | 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.                   |
| <a href="#">.NET core</a>                    | 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.                        |
| <a href="#">Go</a>                           | 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.   |
| <a href="#">Java</a>                         | This quickstart demonstrates how to use Java to connect to an Azure SQL database and then use Transact-SQL statements to query data.  |
| <a href="#">Node.js</a>                      | 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.  |

|                        |   |
|------------------------|---|
| <a href="#">PHP</a>    | 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.  |
| <a href="#">Python</a> | This quickstart demonstrates how to use Python to connect to an Azure SQL database and use Transact-SQL statements to query data.                   |
| <a href="#">Ruby</a>   | 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. |
|                        |   |

## 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.

## Next steps

For connectivity architecture information, see [Azure SQL Database Connectivity Architecture](#).

# Azure SQL Database: Use SQL Server Management Studio to connect and query data

9/24/2018 • 4 minutes to read • [Edit Online](#)

[SQL Server Management Studio \(SSMS\)](#) is an integrated environment for managing any SQL infrastructure, from SQL Server to SQL Database for Microsoft Windows. 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.

## Prerequisites

This quickstart uses as its starting point the resources created in one of these quickstarts:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)

This quickstart also requires that you configure a server-level firewall rule. For a quickstart showing how to do this, see [Create server-level firewall rule](#).

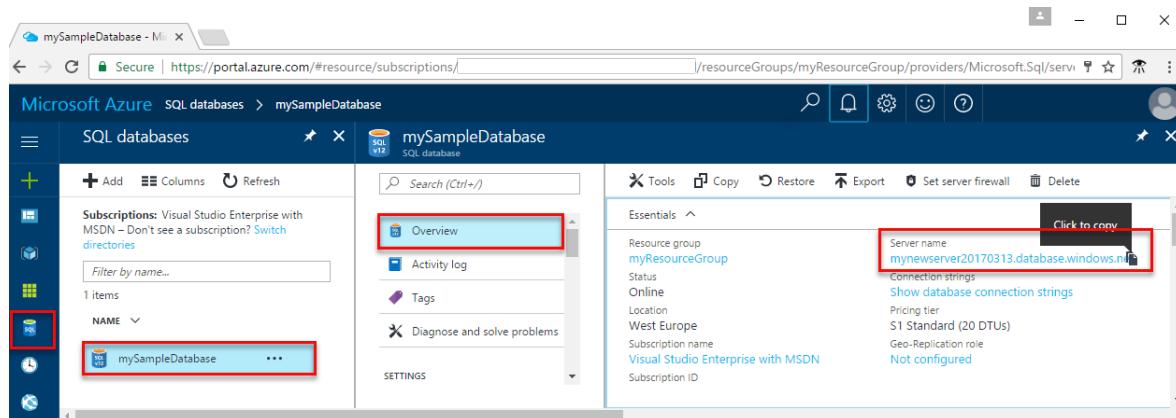
### Install the latest SSMS

Before you start, make sure you have installed the newest version of [SSMS](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forgot your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

# Connect to your database

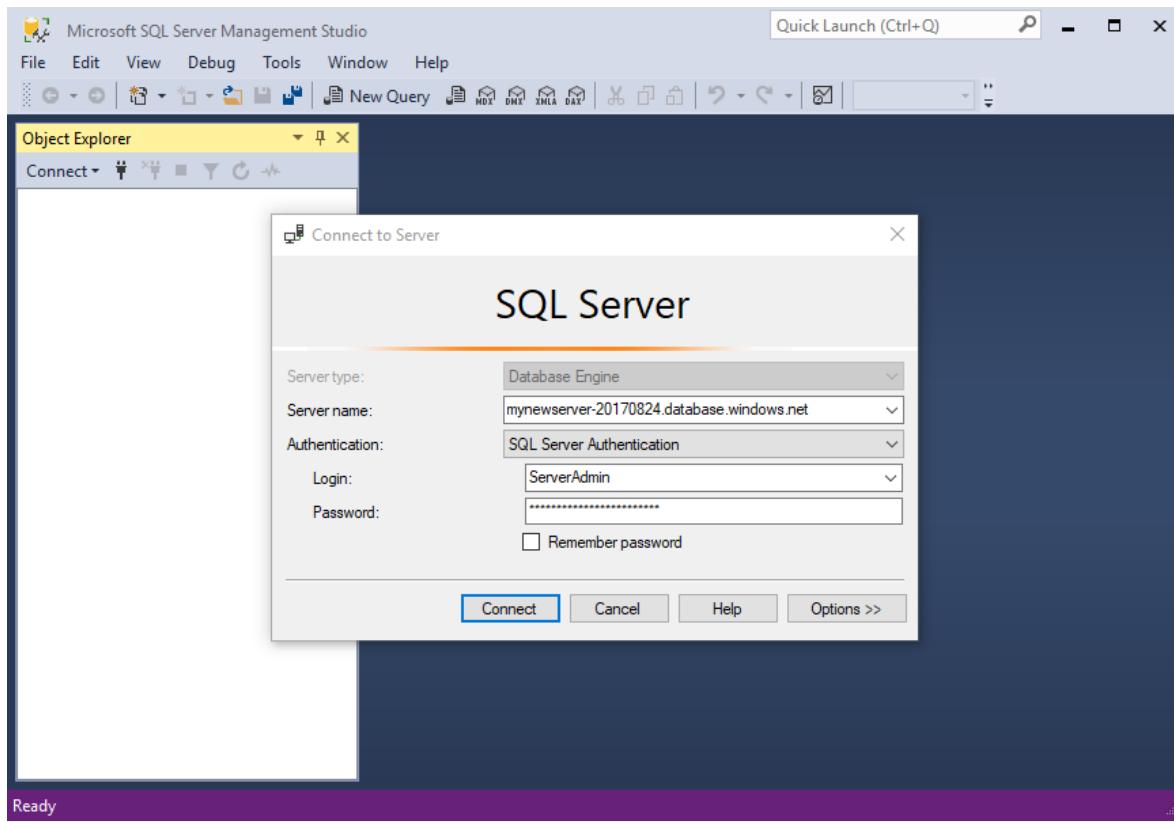
Use SQL Server Management Studio to establish a connection to your Azure SQL Database server.

## IMPORTANT

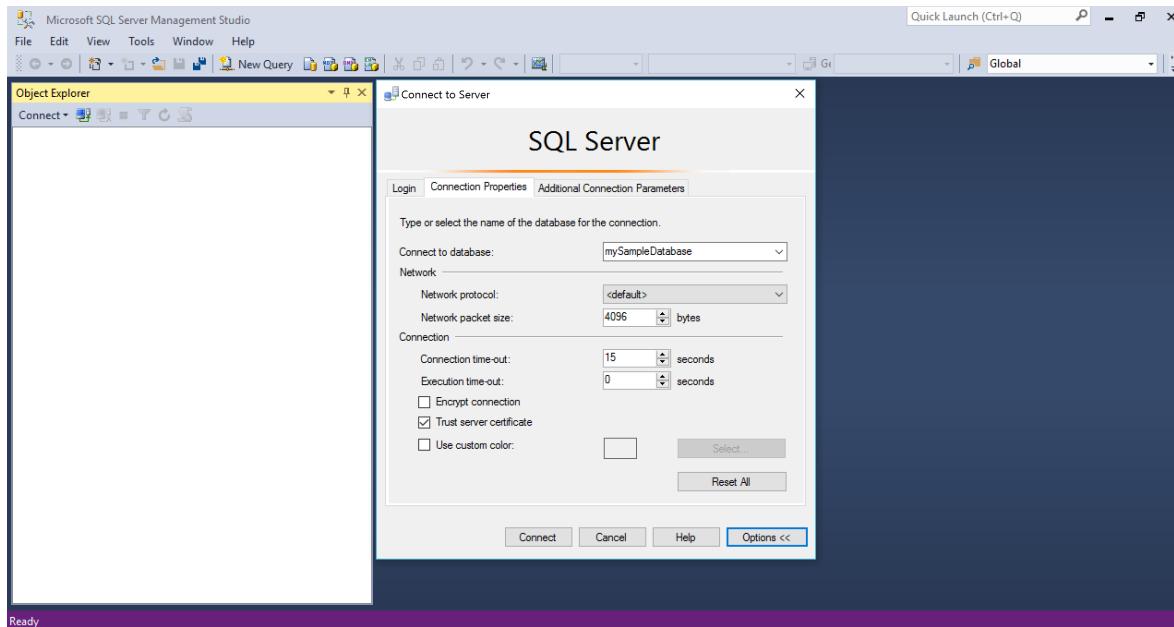
An Azure SQL Database logical server listens on port 1433. If you are attempting to connect to an Azure SQL Database logical server from within a corporate firewall, this port must be open in the corporate firewall for you to successfully connect.

1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

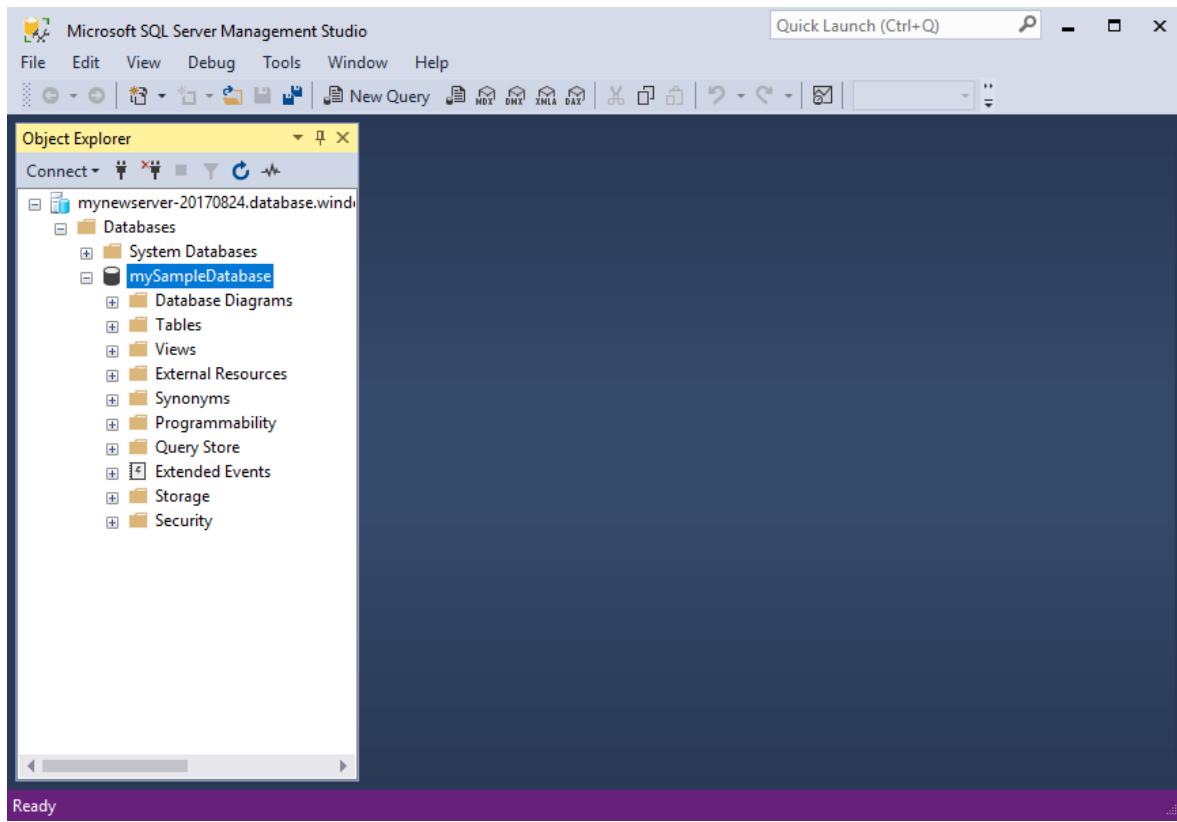
| SETTING               | SUGGESTED VALUE                            | DESCRIPTION  |
|-----------------------|--|--|
| <b>Server type</b>    | Database engine                            | This value is required.  |
| <b>Server name</b>    | The fully qualified server name            | The name should be something like this:<br><b>mynewserver20170313.database.windows.net.</b>  |
| <b>Authentication</b> | SQL Server Authentication                  | SQL Authentication is the only authentication type that we have configured in this tutorial. |
| <b>Login</b>          | The server admin account                   | This is the account that you specified when you created the server.                          |
| <b>Password</b>       | The password for your server admin account | This is 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 **mySampleDatabase** to connect to this database.



4. Click **Connect**. The Object Explorer window opens in SSMS.



5. In Object Explorer, expand **Databases** and then expand **mySampleDatabase** to view the objects in the sample database.

## Query data

Use the following code to query for the top 20 products by category using the **SELECT** Transact-SQL statement.

1. In Object Explorer, right-click **mySampleDatabase** and click **New Query**. A blank query window opens that is connected to your database.
2. In the query window, enter the following 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, click **Execute** to retrieve data from the Product and ProductCategory tables.

The screenshot shows the SSMS interface with the following details:

- Toolbar:** File, Edit, View, Query, Project, Debug, Tools, Window, Help.
- Object Explorer:** Shows the database structure under "mynewserver20170313.database.windows.net". A red box highlights the "mySampleDatabase" node.
- Query Window:** Title bar says "SQLQuery1.sql - mynewserver20170313.database.windows.net.mySampleDatabase (ServerAdmin (...))". The toolbar has an "Execute" button highlighted with a red box.
- Code:** A T-SQL SELECT statement is pasted into the query window:

```
SELECT pc.Name as CategoryName
      , p.Name as ProductName
      , p.Color as Color
      , p.StandardCost as Cost
      , p.ListPrice as Price
      , p.SellStartDate
FROM [SalesLT].[ProductCategory] pc
JOIN [SalesLT].[Product] p
ON pc.productcategoryid = p.productcategoryid;
```
- Results:** A table titled "Results" displays the query results. The table has columns: CategoryName, ProductName, Color, Cost, Price, and SellStartDate. It contains 5 rows of data, all for "Mountain Bikes".
- Status Bar:** Shows "Ready", "Ln 1", "Col 1", "Ch 1", and "INS".

## Insert data

Use the following code to insert a new product into the SalesLT.Product table using the **INSERT** Transact-SQL statement.

1. In the query window, replace the previous query with the following query:

```
INSERT INTO [SalesLT].[Product]
( [Name]
, [ProductNumber]
, [Color]
, [ProductCategoryID]
, [StandardCost]
, [ListPrice]
, [SellStartDate]
)
VALUES
( 'myNewProduct'
,123456789
,'NewColor'
,1
,100
,100
,GETDATE() );
```

2. On the toolbar, click **Execute** to insert a new row in the Product table.

The screenshot shows the SSMS interface. The Object Explorer on the left lists the database structure for 'mynewserver20170313.database.windows.net.mySampleDatabase'. The query window on the right contains the following T-SQL code:

```
INSERT INTO [SalesLT].[Product]
(
    [Name]
    ,[ProductNumber]
    ,[Color]
    ,[ProductCategoryID]
    ,[StandardCost]
    ,[ListPrice]
    ,[SellStartDate]
)
VALUES
(
    'myNewProduct'
    ,123456789
    ,'NewColor'
    ,1
    ,100
    ,100
    ,GETDATE()
);
```

## Update data

Use the following code to update the new product that you previously added using the [UPDATE](#) Transact-SQL statement.

1. In the query window, replace the previous query with the following query:

```
UPDATE [SalesLT].[Product]
SET [ListPrice] = 125
WHERE Name = 'myNewProduct';
```

2. On the toolbar, click **Execute** to update the specified row in the Product table.

The screenshot shows the SSMS interface. The Object Explorer on the left lists the database structure for 'mynewserver20170313.database.windows.net.mySampleDatabase'. The query window on the right contains the following T-SQL code:

```
UPDATE [SalesLT].[Product]
SET [ListPrice] = 125
WHERE Name = 'myNewProduct';
```

The 'Messages' pane below the query window displays the output: '(1 row(s) affected)'.

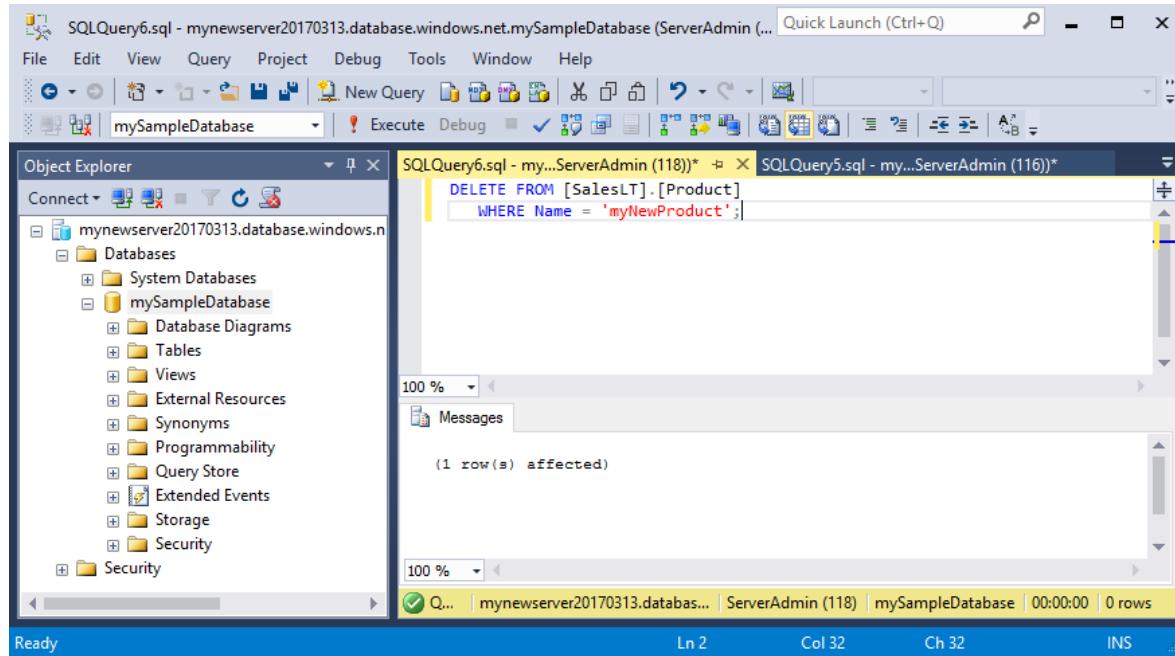
## Delete data

Use the following code to delete the new product that you previously added using the [DELETE](#) Transact-SQL statement.

1. In the query window, replace the previous query with the following query:

```
DELETE FROM [SalesLT].[Product]
WHERE Name = 'myNewProduct';
```

2. On the toolbar, click **Execute** to delete the specified row in the Product table.



## Next steps

- For information about SSMS, see [Use 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](#).

# Azure portal: Use the SQL Query editor to connect and query data

10/30/2018 • 4 minutes to read • [Edit Online](#)

The SQL Query editor is a browser query tool that provides an efficient and lightweight way to execute SQL queries on your Azure SQL Database or Azure SQL Data Warehouse without leaving the 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.

## Prerequisites

This quickstart uses as its starting point the resources created in one of these quickstarts:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)

### NOTE

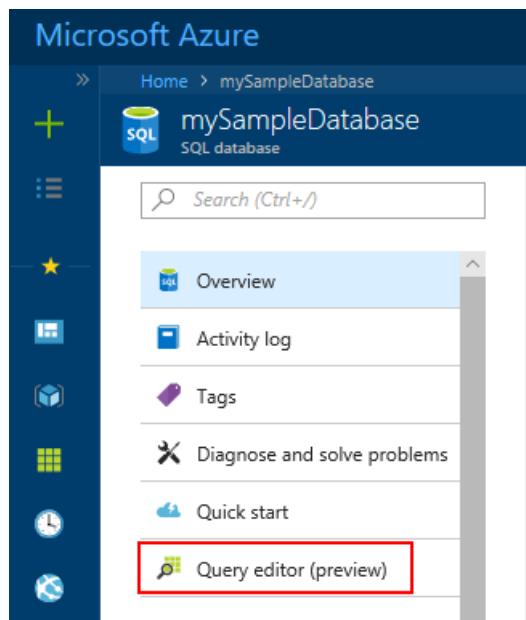
Make sure that the "Allow access to Azure Services" option is set to "ON" in your SQL Server firewall settings. This option gives the SQL Query editor access to your databases and data warehouses.

## Log in to the Azure portal

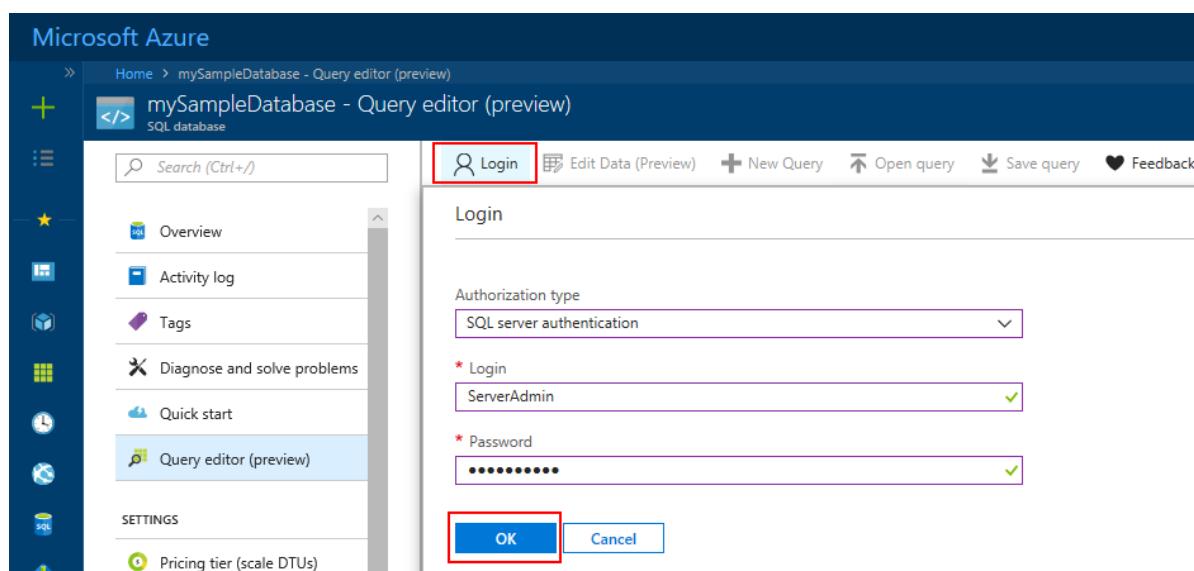
Sign in to the [Azure portal](#).

## Connect using SQL Authentication

1. Click **SQL databases** from the left-hand menu and click the database you would like to query.
2. On the SQL database page for your database, find and click **Query editor (preview)** in the left-hand menu.



3. Click **Login** and then, when prompted, select **SQL Server authentication** and then provide the server admin login and password you provided when creating the database.



4. Click **OK** to login.

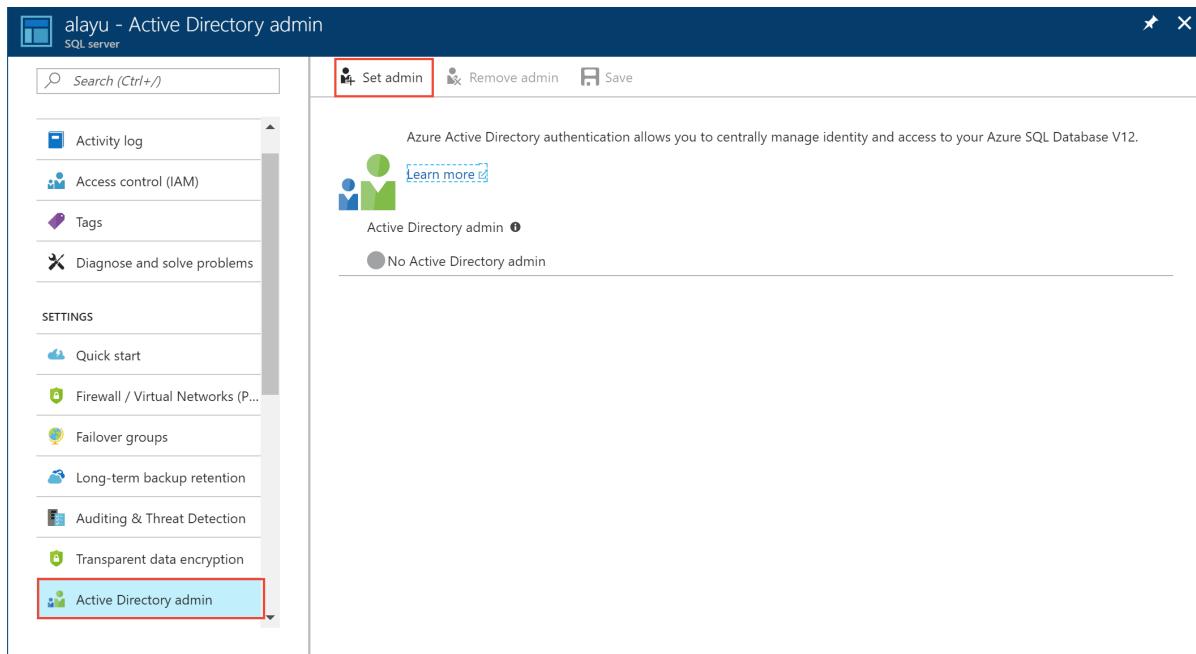
## Connect using Azure AD

Configuring an Active Directory administrator enables you to use a single identity to login to the Azure portal and your SQL database. Follow the steps below to configure an active directory admin for the SQL Server you created.

### NOTE

Email accounts (for example outlook.com, hotmail.com, live.com, gmail.com, yahoo.com) are not yet supported as Active Directory administrators. Make sure to choose a user that was either created natively in the Azure Active Directory, or federated into the Azure Active directory.

1. Select **SQL Servers** from the left-hand menu, and select your SQL Server from the server list.
2. Select the **Active Directory Admin** setting from the settings menu of your SQL Server.
3. In the Active Directory admin blade, click the **Set admin** command, and select the user or group that will be the Active Directory administrator.



4. At the top of the Active Directory admin blade, click the **Save** command to set your Active Directory administrator.

Navigate to the SQL database you would like to query, click **Data explorer (preview)** from the left-hand menu. The Data explorer page opens and automatically connects you to the database.

## Run query using Query Editor

After you are authenticated, type the following query in the Query editor pane to query for 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;
```

Click **Run** and then review the query results in the **Results** pane.

```

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;

```

| 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              |

Query succeeded | 5s

## Insert data using Query Editor

Use the following code to insert a new product into the SalesLT.Product table using the **INSERT** Transact-SQL statement.

- In the query window, replace the previous query with the following query:

```

INSERT INTO [SalesLT].[Product]
(
    [Name]
    , [ProductNumber]
    , [Color]
    , [ProductCategoryID]
    , [StandardCost]
    , [ListPrice]
    , [SellStartDate]
)
VALUES
(
    'myNewProduct'
    ,123456789
    ,'NewColor'
    ,1
    ,100
    ,100
    ,GETDATE() );

```

- On the toolbar, click **Run** to insert a new row in the Product table.

## Update data using Query Editor

Use the following code to update the new product that you previously added using the **UPDATE** Transact-SQL statement.

- In the query window, replace the previous query with the following query:

```
UPDATE [SalesLT].[Product]
SET [ListPrice] = 125
WHERE Name = 'myNewProduct';
```

2. On the toolbar, click **Run** to update the specified row in the Product table.

## Delete data using Query Editor

Use the following code to delete the new product that you previously added using the **DELETE** Transact-SQL statement.

1. In the query window, replace the previous query with the following query:

```
DELETE FROM [SalesLT].[Product]
WHERE Name = 'myNewProduct';
```

2. On the toolbar, click **Run** to delete the specified row in the Product table.

## Query Editor considerations

There are a few things to know when working with the Query editor:

1. Make sure that you have set the "Allow access to Azure Services" option in your Azure SQL Server firewall settings to "ON". This option gives the SQL Query Editor access to your SQL databases and data warehouses.
2. If the SQL server is in a Virtual Network, the Query editor cannot be used to query the databases in that server.
3. Pressing the F5 key will refresh the Query editor page and lose the query that is being worked on. Use the Run button on the toolbar to execute queries.
4. Query editor does not support connecting to master DB
5. There is a 5 minute timeout for query execution.
6. Azure Active Directory Administrator login does not work with accounts that have 2-factor authentication enabled.
7. Email accounts (for example outlook.com, hotmail.com, live.com, gmail.com, yahoo.com) are not yet supported as Active Directory administrators. Make sure to choose a user that was either created natively in the Azure Active Directory, or federated into the Azure Active directory
8. The Query editor only supports cylindrical projection for geography data types.
9. There is no support for IntelliSense for database tables and views. However, the editor does support auto-complete on names that have already been typed.

## Next steps

- To learn about the Transact-SQL supported in Azure SQL databases, see [Transact-SQL differences in SQL database](#).

# Azure SQL Database: Use Visual Studio Code to connect and query data

10/16/2018 • 4 minutes to read • [Edit Online](#)

Visual Studio Code is a graphical code editor for Linux, macOS, and Windows that supports extensions, including the [mssql extension](#) for querying Microsoft SQL Server, Azure SQL Database, and SQL Data Warehouse. 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.

## Prerequisites

This quickstart uses as its starting point the resources created in one of these quickstarts:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)

### Install VS Code

Before you start, make sure you have installed the newest version of [Visual Studio Code](#) and loaded the [mssql extension](#). For installation guidance for the mssql extension, see [Install VS Code](#) and see [mssql for Visual Studio Code](#).

## Configure VS 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.

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.

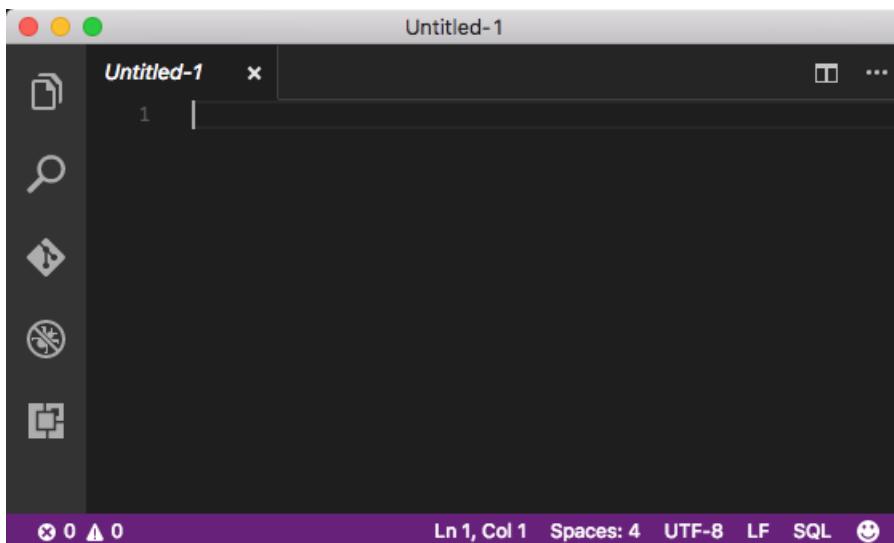
The screenshot shows the Azure portal interface for managing SQL databases. On the left, a sidebar lists 'SQL databases' and shows one item named 'mySampleDatabase'. In the main content area, the 'Overview' tab is selected. The 'Essentials' section displays various database details. The 'Server name' field, which contains the value 'mynewserver20170313.database.windows.net', is highlighted with a red box. To the right of this field is a small button labeled 'Click to copy', also enclosed in a red box.

4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

## Set language mode to SQL

Set the language mode is set to **SQL** in Visual Studio Code to enable mssql commands and T-SQL IntelliSense.

1. Open a new Visual Studio Code window.
2. Click **Plain Text** in the lower right-hand corner of the status bar.
3. In the **Select language mode** drop-down menu that opens, type **SQL**, and then press **ENTER** to set the language mode to SQL.



## Connect to your database

Use Visual Studio Code to establish a connection to your Azure SQL Database server.

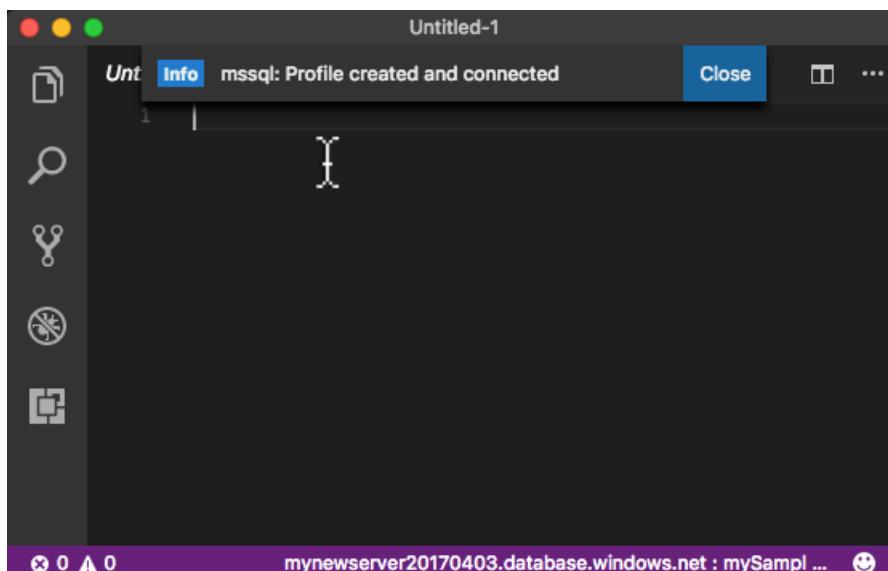
### IMPORTANT

Before continuing, make sure that you have your server, database, and login 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 connection profile.

1. In VS Code, press **CTRL+SHIFT+P** (or **F1**) to open the Command Palette.
2. Type **sqlcon** and press **ENTER**.
3. Press **ENTER** to select **Create Connection Profile**. This creates a connection profile for your SQL Server instance.
4. Follow the prompts to specify the connection properties for the new connection profile. After specifying each value, press **ENTER** to continue.

| SETTING                       | SUGGESTED VALUE                                    | DESCRIPTION  |
|-------------------------------|--|--|
| **Server name                 | The fully qualified server name                    | The name should be something like this:<br><b>mynewserver20170313.database.windows.net</b> . |
| Database name                 | mySampleDatabase                                   | The name of the database to which to connect.  |
| Authentication                | SQL Login  | SQL Authentication is the only authentication type that we have configured in this tutorial. |
| User name                     | The server admin account                           | This is the account that you specified when you created the server.                          |
| Password (SQL Login)          | The password for your server admin account         | This is the password that you specified when you created 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<br><b>mySampleDatabase</b> | A saved profile name speeds your connection on subsequent logins.                            |

5. Press the **ESC** key to close the info message that informs you that the profile is created and connected.
6. Verify your connection in the status bar.



## Query data

Use the following code to query for the top 20 products by category using the **SELECT** Transact-SQL statement.

1. In the **Editor** window, enter the following query in the empty query window:

```
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 retrieve data from the Product and ProductCategory tables.

The screenshot shows the SSMS interface with two windows. On the left is the 'Untitled-1' query editor containing the T-SQL code for selecting products by category. On the right is the 'Results: Untitled-1' window displaying the output of the query, which shows multiple rows of data where the CategoryName is 'Mountain Bikes' and the ProductName is 'Mountain-100...'. Below the results, the message '(295 rows affected)' and 'Total execution time: 00:00:00.367' are visible. The status bar at the bottom indicates the current line (Ln 4), column (Col 47), spaces (Spaces: 4), and encoding (UTF-8).

```
SELECT pc.Name as CategoryName, p.name as ProductName
FROM [SalesLT].[ProductCategory] pc
JOIN [SalesLT].[Product] p
ON pc.productcategoryid = p.productcategoryid;
```

| 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... |
| 8 Mountain Bikes   | Mountain-100... |
| 9 Mountain Bikes   | Mountain-100... |
| 10 Mountain Bikes  | Mountain-100... |
| 11 Mountain Bikes  | Mountain-100... |
| 12 Mountain Bikes  | Mountain-100... |
| 13 Mountain Bikes  | Mountain-100... |
| 14 Mountain Bikes  | Mountain-100... |
| 15 Mountain Bikes  | Mountain-100... |
| 16 Mountain Bikes  | Mountain-100... |
| 17 Mountain Bikes  | Mountain-100... |
| 18 Mountain Bikes  | Mountain-100... |
| 19 Mountain Bikes  | Mountain-100... |
| 20 Mountain Bikes  | Mountain-100... |
| 21 Mountain Bikes  | Mountain-100... |
| 22 Mountain Bikes  | Mountain-100... |
| 23 Mountain Bikes  | Mountain-100... |
| 24 Mountain Bikes  | Mountain-100... |
| 25 Mountain Bikes  | Mountain-100... |
| 26 Mountain Bikes  | Mountain-100... |
| 27 Mountain Bikes  | Mountain-100... |
| 28 Mountain Bikes  | Mountain-100... |
| 29 Mountain Bikes  | Mountain-100... |
| 30 Mountain Bikes  | Mountain-100... |
| 31 Mountain Bikes  | Mountain-100... |
| 32 Mountain Bikes  | Mountain-100... |
| 33 Mountain Bikes  | Mountain-100... |
| 34 Mountain Bikes  | Mountain-100... |
| 35 Mountain Bikes  | Mountain-100... |
| 36 Mountain Bikes  | Mountain-100... |
| 37 Mountain Bikes  | Mountain-100... |
| 38 Mountain Bikes  | Mountain-100... |
| 39 Mountain Bikes  | Mountain-100... |
| 40 Mountain Bikes  | Mountain-100... |
| 41 Mountain Bikes  | Mountain-100... |
| 42 Mountain Bikes  | Mountain-100... |
| 43 Mountain Bikes  | Mountain-100... |
| 44 Mountain Bikes  | Mountain-100... |
| 45 Mountain Bikes  | Mountain-100... |
| 46 Mountain Bikes  | Mountain-100... |
| 47 Mountain Bikes  | Mountain-100... |
| 48 Mountain Bikes  | Mountain-100... |
| 49 Mountain Bikes  | Mountain-100... |
| 50 Mountain Bikes  | Mountain-100... |
| 51 Mountain Bikes  | Mountain-100... |
| 52 Mountain Bikes  | Mountain-100... |
| 53 Mountain Bikes  | Mountain-100... |
| 54 Mountain Bikes  | Mountain-100... |
| 55 Mountain Bikes  | Mountain-100... |
| 56 Mountain Bikes  | Mountain-100... |
| 57 Mountain Bikes  | Mountain-100... |
| 58 Mountain Bikes  | Mountain-100... |
| 59 Mountain Bikes  | Mountain-100... |
| 60 Mountain Bikes  | Mountain-100... |
| 61 Mountain Bikes  | Mountain-100... |
| 62 Mountain Bikes  | Mountain-100... |
| 63 Mountain Bikes  | Mountain-100... |
| 64 Mountain Bikes  | Mountain-100... |
| 65 Mountain Bikes  | Mountain-100... |
| 66 Mountain Bikes  | Mountain-100... |
| 67 Mountain Bikes  | Mountain-100... |
| 68 Mountain Bikes  | Mountain-100... |
| 69 Mountain Bikes  | Mountain-100... |
| 70 Mountain Bikes  | Mountain-100... |
| 71 Mountain Bikes  | Mountain-100... |
| 72 Mountain Bikes  | Mountain-100... |
| 73 Mountain Bikes  | Mountain-100... |
| 74 Mountain Bikes  | Mountain-100... |
| 75 Mountain Bikes  | Mountain-100... |
| 76 Mountain Bikes  | Mountain-100... |
| 77 Mountain Bikes  | Mountain-100... |
| 78 Mountain Bikes  | Mountain-100... |
| 79 Mountain Bikes  | Mountain-100... |
| 80 Mountain Bikes  | Mountain-100... |
| 81 Mountain Bikes  | Mountain-100... |
| 82 Mountain Bikes  | Mountain-100... |
| 83 Mountain Bikes  | Mountain-100... |
| 84 Mountain Bikes  | Mountain-100... |
| 85 Mountain Bikes  | Mountain-100... |
| 86 Mountain Bikes  | Mountain-100... |
| 87 Mountain Bikes  | Mountain-100... |
| 88 Mountain Bikes  | Mountain-100... |
| 89 Mountain Bikes  | Mountain-100... |
| 90 Mountain Bikes  | Mountain-100... |
| 91 Mountain Bikes  | Mountain-100... |
| 92 Mountain Bikes  | Mountain-100... |
| 93 Mountain Bikes  | Mountain-100... |
| 94 Mountain Bikes  | Mountain-100... |
| 95 Mountain Bikes  | Mountain-100... |
| 96 Mountain Bikes  | Mountain-100... |
| 97 Mountain Bikes  | Mountain-100... |
| 98 Mountain Bikes  | Mountain-100... |
| 99 Mountain Bikes  | Mountain-100... |
| 100 Mountain Bikes | Mountain-100... |
| 101 Mountain Bikes | Mountain-100... |
| 102 Mountain Bikes | Mountain-100... |
| 103 Mountain Bikes | Mountain-100... |
| 104 Mountain Bikes | Mountain-100... |
| 105 Mountain Bikes | Mountain-100... |
| 106 Mountain Bikes | Mountain-100... |
| 107 Mountain Bikes | Mountain-100... |
| 108 Mountain Bikes | Mountain-100... |
| 109 Mountain Bikes | Mountain-100... |
| 110 Mountain Bikes | Mountain-100... |
| 111 Mountain Bikes | Mountain-100... |
| 112 Mountain Bikes | Mountain-100... |
| 113 Mountain Bikes | Mountain-100... |
| 114 Mountain Bikes | Mountain-100... |
| 115 Mountain Bikes | Mountain-100... |
| 116 Mountain Bikes | Mountain-100... |
| 117 Mountain Bikes | Mountain-100... |
| 118 Mountain Bikes | Mountain-100... |
| 119 Mountain Bikes | Mountain-100... |
| 120 Mountain Bikes | Mountain-100... |
| 121 Mountain Bikes | Mountain-100... |
| 122 Mountain Bikes | Mountain-100... |
| 123 Mountain Bikes | Mountain-100... |
| 124 Mountain Bikes | Mountain-100... |
| 125 Mountain Bikes | Mountain-100... |
| 126 Mountain Bikes | Mountain-100... |
| 127 Mountain Bikes | Mountain-100... |
| 128 Mountain Bikes | Mountain-100... |
| 129 Mountain Bikes | Mountain-100... |
| 130 Mountain Bikes | Mountain-100... |
| 131 Mountain Bikes | Mountain-100... |
| 132 Mountain Bikes | Mountain-100... |
| 133 Mountain Bikes | Mountain-100... |
| 134 Mountain Bikes | Mountain-100... |
| 135 Mountain Bikes | Mountain-100... |
| 136 Mountain Bikes | Mountain-100... |
| 137 Mountain Bikes | Mountain-100... |
| 138 Mountain Bikes | Mountain-100... |
| 139 Mountain Bikes | Mountain-100... |
| 140 Mountain Bikes | Mountain-100... |
| 141 Mountain Bikes | Mountain-100... |
| 142 Mountain Bikes | Mountain-100... |
| 143 Mountain Bikes | Mountain-100... |
| 144 Mountain Bikes | Mountain-100... |
| 145 Mountain Bikes | Mountain-100... |
| 146 Mountain Bikes | Mountain-100... |
| 147 Mountain Bikes | Mountain-100... |
| 148 Mountain Bikes | Mountain-100... |
| 149 Mountain Bikes | Mountain-100... |
| 150 Mountain Bikes | Mountain-100... |
| 151 Mountain Bikes | Mountain-100... |
| 152 Mountain Bikes | Mountain-100... |
| 153 Mountain Bikes | Mountain-100... |
| 154 Mountain Bikes | Mountain-100... |
| 155 Mountain Bikes | Mountain-100... |
| 156 Mountain Bikes | Mountain-100... |
| 157 Mountain Bikes | Mountain-100... |
| 158 Mountain Bikes | Mountain-100... |
| 159 Mountain Bikes | Mountain-100... |
| 160 Mountain Bikes | Mountain-100... |
| 161 Mountain Bikes | Mountain-100... |
| 162 Mountain Bikes | Mountain-100... |
| 163 Mountain Bikes | Mountain-100... |
| 164 Mountain Bikes | Mountain-100... |
| 165 Mountain Bikes | Mountain-100... |
| 166 Mountain Bikes | Mountain-100... |
| 167 Mountain Bikes | Mountain-100... |
| 168 Mountain Bikes | Mountain-100... |
| 169 Mountain Bikes | Mountain-100... |
| 170 Mountain Bikes | Mountain-100... |
| 171 Mountain Bikes | Mountain-100... |
| 172 Mountain Bikes | Mountain-100... |
| 173 Mountain Bikes | Mountain-100... |
| 174 Mountain Bikes | Mountain-100... |
| 175 Mountain Bikes | Mountain-100... |
| 176 Mountain Bikes | Mountain-100... |
| 177 Mountain Bikes | Mountain-100... |
| 178 Mountain Bikes | Mountain-100... |
| 179 Mountain Bikes | Mountain-100... |
| 180 Mountain Bikes | Mountain-100... |
| 181 Mountain Bikes | Mountain-100... |
| 182 Mountain Bikes | Mountain-100... |
| 183 Mountain Bikes | Mountain-100... |
| 184 Mountain Bikes | Mountain-100... |
| 185 Mountain Bikes | Mountain-100... |
| 186 Mountain Bikes | Mountain-100... |
| 187 Mountain Bikes | Mountain-100... |
| 188 Mountain Bikes | Mountain-100... |
| 189 Mountain Bikes | Mountain-100... |
| 190 Mountain Bikes | Mountain-100... |
| 191 Mountain Bikes | Mountain-100... |
| 192 Mountain Bikes | Mountain-100... |
| 193 Mountain Bikes | Mountain-100... |
| 194 Mountain Bikes | Mountain-100... |
| 195 Mountain Bikes | Mountain-100... |
| 196 Mountain Bikes | Mountain-100... |
| 197 Mountain Bikes | Mountain-100... |
| 198 Mountain Bikes | Mountain-100... |
| 199 Mountain Bikes | Mountain-100... |
| 200 Mountain Bikes | Mountain-100... |

## Insert data

Use the following code to insert a new product into the SalesLT.Product table using the **INSERT** Transact-SQL statement.

1. In the **Editor** window, delete the previous query and enter the following query:

```
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

Use the following code to update the new product that you previously added using the [UPDATE](#) Transact-SQL statement.

1. In the **Editor** window, delete the previous query and enter the following query:

```
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

Use the following code to delete the new product that you previously added using the [DELETE](#) Transact-SQL statement.

1. In the **Editor** window, delete the previous query and enter the following query:

```
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 [Connect and query with SSMS](#).
- To connect and query using the Azure portal, see [Connect and query with the Azure portal SQL query editor](#).
- For an MSDN magazine article on using Visual Studio Code, see [Create a database IDE with MSSQL extension blog post](#).

# Use .NET (C#) with Visual Studio to connect and query an Azure SQL database

9/24/2018 • 3 minutes to read • [Edit Online](#)

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.

## Prerequisites

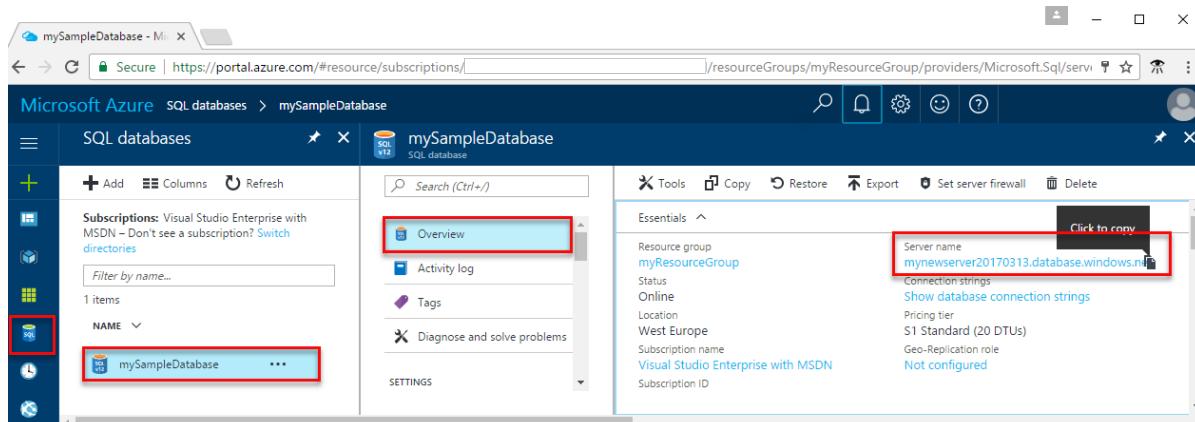
To complete this quickstart, make sure you have the following:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- An installation of [Visual Studio Community 2017](#), [Visual Studio Professional 2017](#), or [Visual Studio Enterprise 2017](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

### For ADO.NET

1. Continue by clicking **Show database connection strings**.
2. Review the complete **ADO.NET** connection string.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with 'Database connection strings' selected. The main area shows a 'mySampleDatabase' resource with various details like server name, location, and monitoring. On the right, a detailed view of the 'Database connection strings' for 'mySampleDatabase' is shown. The 'ADO.NET' tab is active, and the 'ADO.NET (SQL authentication)' section displays a connection string template. A red box highlights this template:

```
Server=tcp:mynewserver20170327.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;
```

#### IMPORTANT

You must have a firewall rule in place for the public IP address of the computer on which you perform this tutorial. If you are on a different computer or have a different public IP address, create a [server-level firewall rule using the Azure portal](#).

## Create a new Visual Studio project

1. In Visual Studio, choose **File, New, Project**.
2. In the **New Project** dialog, and expand **Visual C#**.
3. Select **Console App** and enter **sql/test** for the project name.
4. Click **OK** to create and open the new project in Visual Studio
5. In Solution Explorer, right-click **sqltest** and click **Manage NuGet Packages**.
6. On the **Browse** tab, search for **System.Data.SqlClient** and, when found, select it.
7. In the **System.Data.SqlClient** page, click **Install**.
8. When the install completes, review the changes and then click **OK** to close the **Preview** window.
9. If a **License Acceptance** window appears, click **I Accept**.

## Insert code to query SQL database

1. Switch to (or open if necessary) **Program.cs**
2. Replace the contents of **Program.cs** with the following code and add the appropriate values for your server, database, user, and password.

```

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_user";
                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.ReadLine();
        }
    }
}

```

## Run the code

1. Press **F5** to run the application.
2. Verify that the top 20 rows are returned and then close the application 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](#)

# Use .NET Core (C#) to query an Azure SQL database

9/24/2018 • 2 minutes to read • [Edit Online](#)

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.

## Prerequisites

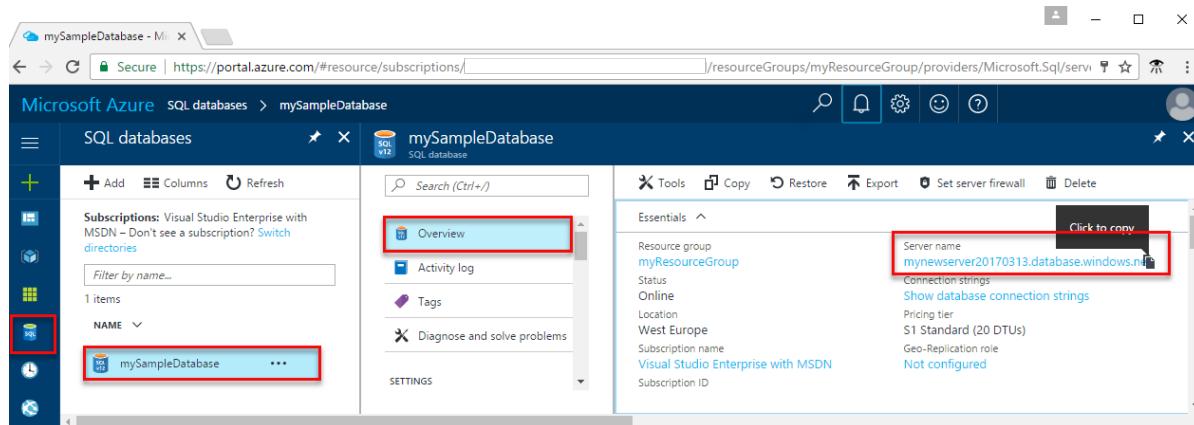
To complete this quickstart, make sure you have the following:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed [.NET Core for your operating system](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

### For ADO.NET

1. Continue by clicking **Show database connection strings**.
2. Review the complete **ADO.NET** connection string.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with icons for tools like Copy, Restore, Export, Set server firewall, and Delete. The main area shows 'Database connection strings' for a database named 'mySampleDatabase'. The 'ADO.NET' tab is selected, showing the connection string configuration. A red box highlights the connection string details:

```
Server=tcp:mynewserver20170327.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;
```

### IMPORTANT

You must have a firewall rule in place for the public IP address of the computer on which you perform this tutorial. If you are on a different computer or have a different public IP address, create a [server-level firewall rule using the Azure portal](#).

## Create a new .NET project

1. Open a command prompt and create a folder named *sqltest*. Navigate to the folder you created and run the following command:

```
dotnet new console
```

2. Open **sqltest.csproj** with your favorite text editor and add System.Data.SqlClient as a dependency using the following code:

```
<ItemGroup>
  <PackageReference Include="System.Data.SqlClient" Version="4.4.0" />
</ItemGroup>
```

## Insert code to query SQL database

1. In your development environment or favorite text editor open **Program.cs**
2. Replace the contents with the following code and add the appropriate values for your server, database, user, and password.

```

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_user";
                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.ReadLine();
        }
    }
}

```

## Run the code

- At the command prompt, run the following commands:

```

dotnet restore
dotnet run

```

- Verify that the top 20 rows are returned and then 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 your first Azure SQL database using .NET.](#)
- For more information about .NET, see [.NET documentation](#).

# Use ActiveDirectoryInteractive mode to connect to Azure SQL Database

9/24/2018 • 8 minutes to read • [Edit Online](#)

This article provides a runnable C# code example that connects to your Microsoft Azure SQL Database. The C# program uses the interactive mode of authentication, which supports Azure AD multi-factor authentication (MFA). For instance, a connection attempt can include a verification code being sent to your mobile phone.

For more information about MFA support for SQL tools, see [Azure Active Directory support in SQL Server Data Tools \(SSDT\)](#).

## SqlAuthenticationMethod .ActiveDirectoryInteractive enum value

Starting in .NET Framework version 4.7.2, the enum **SqlAuthenticationMethod** has a new value

**.ActiveDirectoryInteractive**. When used by a client C# program, this enum value directs the system to use the Azure AD Interactive mode supporting MFA to authenticate to Azure SQL Database. The user who runs the program then sees the following dialogs:

1. A dialog that displays an Azure AD user name, and that asks for the password of the Azure AD user.
  - This dialog is not displayed if no password is needed. No password is needed if the user's domain is federated with Azure AD.If MFA is imposed on the user by the policy set in Azure AD, the following dialogs are displayed next.
2. Only the very first time the user experiences the MFA scenario, the system displays an additional dialog. The dialog asks for a mobile phone number to which text messages will be sent. Each message provides the *verification code* that the user must enter into the next dialog.
3. Another dialog that asks for the MFA verification code, which the system has sent to a mobile phone.

For information about how to configure Azure AD to require MFA, see [Getting started with Azure Multi-Factor Authentication in the cloud](#).

For screenshots of these dialogs, see [Configure multi-factor authentication for SQL Server Management Studio and Azure AD](#).

### TIP

Our general search page for all kinds of .NET Framework APIs is available at the following link to our handy **.NET API Browser** tool:

<https://docs.microsoft.com/dotnet/api/>

By adding the type name to the optional appended **?term=** parameter, the search page can have our result ready and waiting for us:

<https://docs.microsoft.com/dotnet/api/?term=SqlAuthenticationMethod>

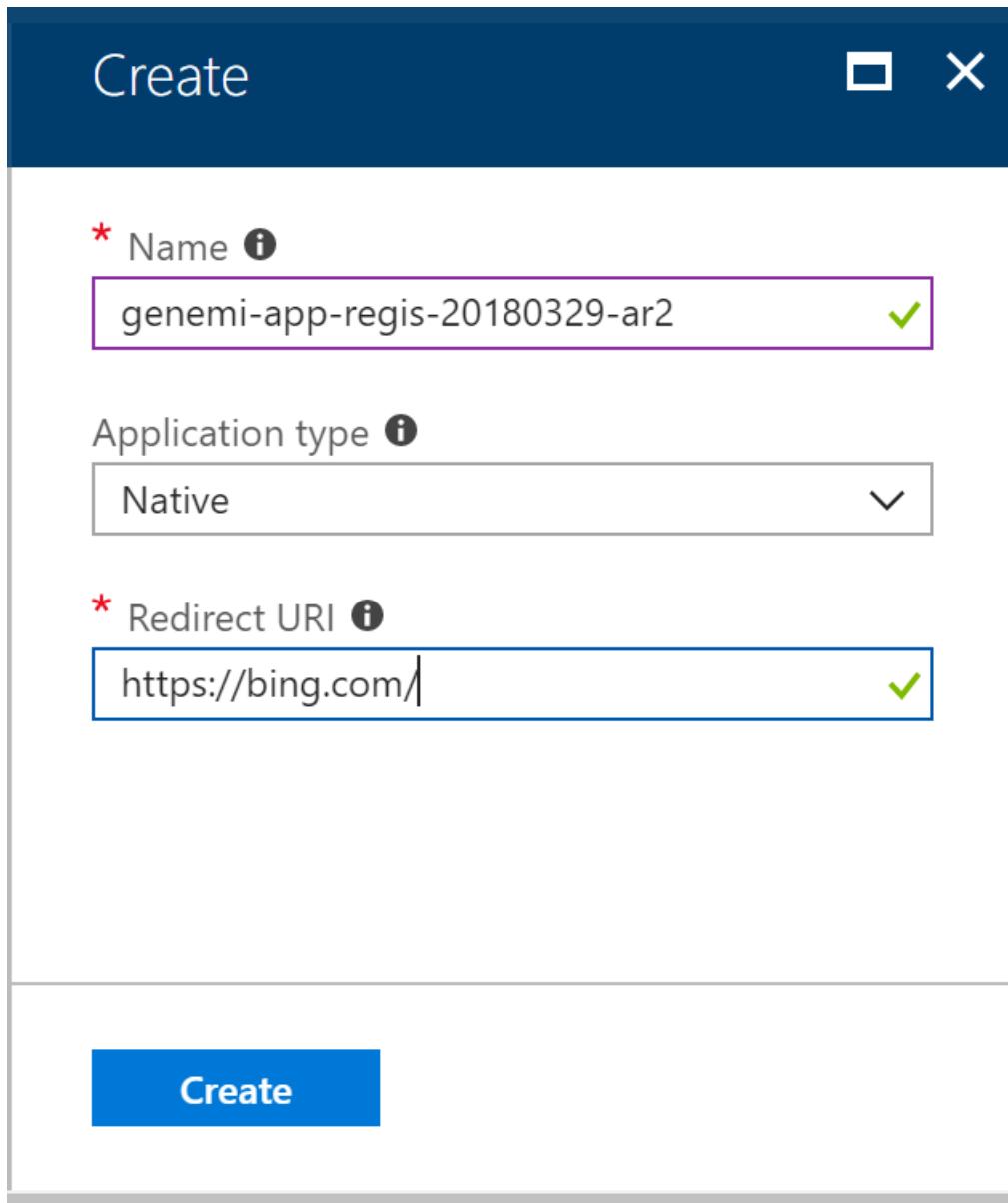
## Preparations for C#, by using the Azure portal

We assume that you already have an [Azure SQL Database server created](#) and available.

## A. Create an app registration

To use Azure AD authentication, your C# client program must supply a GUID as a *clientId* when your program attempts to connect. Completing an app registration generates and displays the GUID in the Azure portal, labeled as **Application ID**. The navigation steps are as follows:

1. Azure portal > **Azure Active Directory** > **App registration**

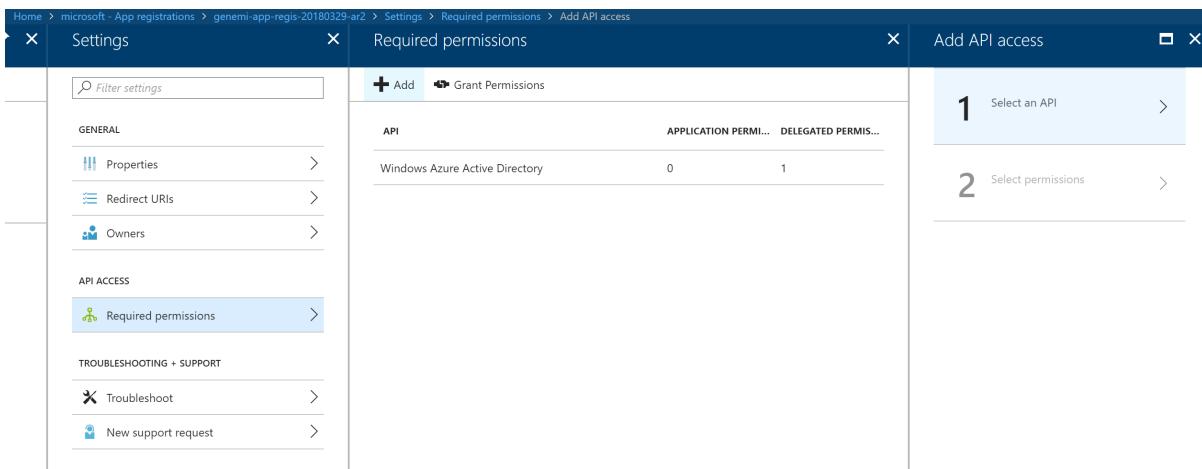


2. The **Application ID** value is generated and displayed.

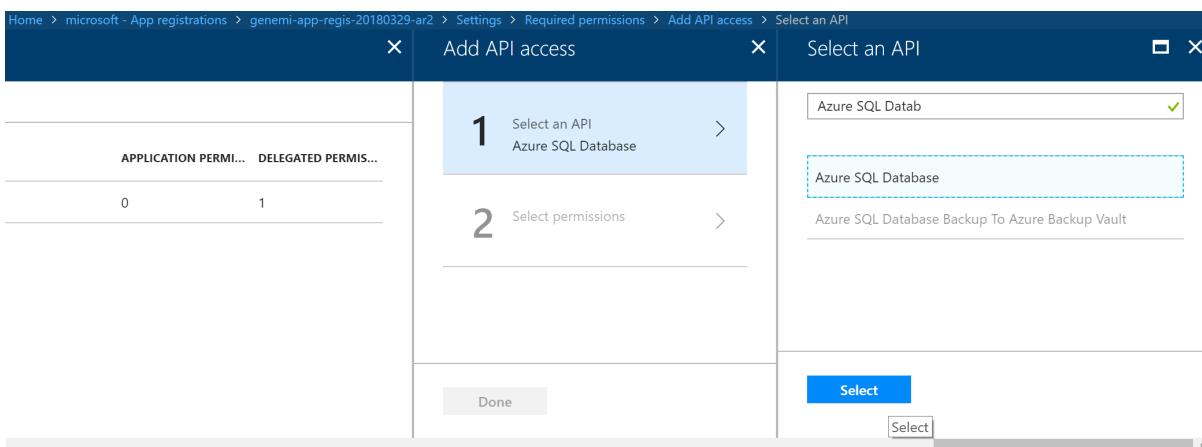
The screenshot shows the details page for the registered app. It lists the following information:

|                  |                               |  |                                      |
|------------------|-------------------------------|--|--------------------------------------|
| Display name     | genemi-app-regis-20180329-ar2 | Application ID                         | 46097988-80d4-4f4f-ac55-39ba7a9672eb |
| Application type | Native                        | Object ID                              | 5899f9f1-b07e-4853-9653-7f143e6ba564 |
| Home page        | --                            | Managed application in local directory | genemi-app-regis-20180329-ar2        |

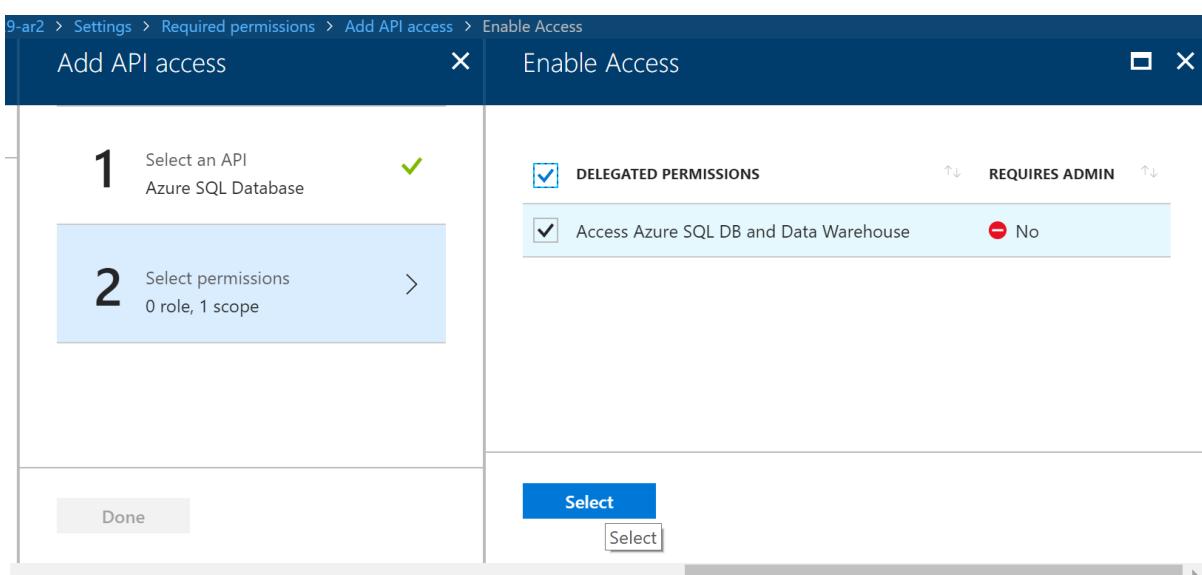
3. **Registered app** > **Settings** > **Required permissions** > **Add**



#### 4. Required permissions > Add API access > Select an API > Azure SQL Database



#### 5. API access > Select permissions > Delegated permissions



#### B. Set Azure AD admin on your SQL Database server

Each Azure SQL Database server has its own SQL logical server of Azure AD. For our C# scenario, you must set an Azure AD administrator for your Azure SQL server.

##### 1. SQL Server > 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](#), in its section **Provision an Azure Active Directory administrator for your Azure SQL Database server**.

### C. Prepare an Azure AD user to connect to a specific database

In the Azure AD that is specific to your Azure SQL Database server, you can add a user who shall have access to a particular database.

For more information, see [Use Azure Active Directory Authentication for authentication with SQL Database, Managed Instance, or SQL Data Warehouse](#).

### D. Add a non-admin user to Azure AD

The Azure AD admin of SQL Database server can be used to connect to your SQL Database server. However, a more general case is to add a non-admin user to the Azure AD. When the non-admin user is used to connect, the MFA sequence is invoked if MFA is imposed on this user by Azure AD.

## Azure Active Directory Authentication Library (ADAL)

The C# program relies on the namespace **Microsoft.IdentityModel.Clients.ActiveDirectory**. The classes for this namespace are in the assembly by the same name.

- Use NuGet to download and install the ADAL assembly.
  - <https://www.nuget.org/packages/Microsoft.IdentityModel.Clients.ActiveDirectory/>
- Add a reference to the assembly, to support a compile of the C# program.

## SqlAuthenticationMethod enum

One namespaces that the C# example relies on is **System.Data.SqlClient**. Of special interest is the enum **SqlAuthenticationMethod**. This enum has the following values:

- **SqlAuthenticationMethod.ActiveDirectory \*Interactive\***: Use this with an Azure AD user name, to achieve multi-factor authentication MFA.
  - This value is the focus of the present article. It produces an interactive experience by displaying dialogs for the user password, and then for MFA validation if MFA is imposed on this user.
  - This value is available starting with .NET Framework version 4.7.2.
- **SqlAuthenticationMethod.ActiveDirectory \*Integrated\***: Use this for a \*federated account. For a federated account, the user name is known to the Windows domain. This method does not support MFA.
- **SqlAuthenticationMethod.ActiveDirectory \*Password\***: Use this for authentication that requires an Azure AD user and the user's password. Azure SQL Database performs the authentication. This method does not support MFA.

## Prepare C# parameter values from the Azure portal

For a successful run of the C# program, you must assign the proper values to the following static fields. These static fields act like parameters into the program. The fields are shown here with pretend values. Also shown are the locations in the Azure portal from where you can obtain the proper values:

| STATIC FIELD NAME    | PRETEND VALUE                                | WHERE IN AZURE PORTAL                                       |
|----------------------|--|---|
| Az_SQLDB_svrName     | "my-favorite-sqldb-svr.database.windows.net" | <b>SQL servers &gt; Filter by name</b>                      |
| AzureAD_UserID       | "user9@abc.onmicrosoft.com"                  | <b>Azure Active Directory &gt; User &gt; New guest user</b> |
| Initial_DatabaseName | "master"                                     | <b>SQL servers &gt; SQL databases</b>                       |

| STATIC FIELD NAME   | PRETEND VALUE   | WHERE IN AZURE PORTAL  |
|---------------------|---|--|
| ClientApplicationID | "a94f9c62-97fe-4d19-b06d-111111111111"                          | Azure Active Directory > App registrations<br>> Search by name > Application ID  |
| RedirectUri         | new Uri( " <a href="https://bing.com/">https://bing.com/</a> ") | Azure Active Directory > App registrations<br>> Search by name > [Your-App-regis] ><br>Settings > RedirectURIs<br><br>For this article, any valid value is fine for RedirectUri. The value is not really used in our case. |
|                     |   |  |

Depending on your particular scenario, you might not need values all the parameters in the preceding table.

## Run SSMS to verify

It is helpful to run SQL Server Management Studio (SSMS) before running the C# program. The SSMS run verifies that various configurations are correct. Then any failure of the C# program can be narrowed to just its source code.

### Verify SQL Database firewall IP addresses

Run SSMS from the same computer, in the same building, that you will later run the C# program. You can use whichever **Authentication** mode you feel is the easiest. If there is any indication that the database server firewall is not accepting your IP address, you can fix that as shown in [Azure SQL Database server-level and database-level firewall rules](#).

### Verify multi-factor authentication (MFA) for Azure AD

Run SSMS again, this time with **Authentication** set to **Active Directory - Universal with MFA support**. For this option you must have SSMS version 17.5 or later.

For more information, see [Configure multi-factor authentication for SSMS and Azure AD](#).

## C# code example

To compile this C# example, you must add a reference to the DLL assembly named **Microsoft.IdentityModel.Clients.ActiveDirectory**.

### Reference documentation

- **System.Data.SqlClient** namespace:
  - Search: <https://docs.microsoft.com/dotnet/api/?term=System.Data.SqlClient>
  - Direct: <System.Data.Client>
- **Microsoft.IdentityModel.Clients.ActiveDirectory** namespace:
  - Search: <https://docs.microsoft.com/dotnet/api/?term=Microsoft.IdentityModel.Clients.ActiveDirectory>
  - Direct: <Microsoft.IdentityModel.Clients.ActiveDirectory>

### C# source code, in two parts

```

using System; // C# , part 1 of 2.

// Add a reference to assembly: Microsoft.IdentityModel.Clients.ActiveDirectory.dll
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 ADInteractive5
{
    class Program
    {
        // ASSIGN YOUR VALUES TO THESE STATIC FIELDS !!
        static public string Az_SQLDB_svrName = "<YOUR VALUE HERE>";
        static public string AzureAD_UserID = "<YOUR VALUE HERE>";
        static public string Initial_DatabaseName = "master";
        // Some scenarios do not need values for the following two fields:
        static public readonly string ClientApplicationID = "<YOUR VALUE HERE>";
        static public readonly Uri RedirectUri = new Uri("<YOUR VALUE HERE>");

        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();
                if (sqlConnection.State == DA.ConnectionState.Open)
                {
                    var rdr = cmd.ExecuteReader();
                    var msg = new TX.StringBuilder();
                    while (rdr.Read())
                    {
                        msg.Append(rdr[0].ToString());
                    }
                    Console.WriteLine(msg.ToString());
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}

```

```

        }
        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 .

```

### Second half of the C# program

For better visual display, the C# program is split into two code blocks. To run the program, paste the two code blocks together.

```

// C# , part 2 of 2 , to concatenate below part 1.

/// <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.UserIdentity(
            parameters.UserId,
            AD.UserIdentityType.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.

```

#### Actual test output from C#

```

[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

- [Get-AzureRmSqlServerActiveDirectoryAdministrator](#)

# Use Go to query an Azure SQL database

9/24/2018 • 5 minutes to read • [Edit Online](#)

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.

## Prerequisites

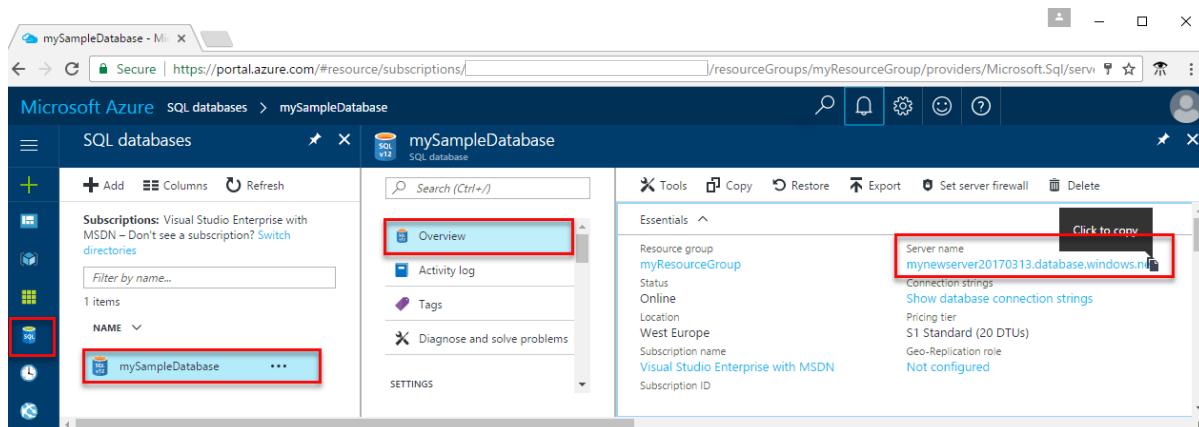
To complete this quickstart, make sure you have the following prerequisites:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed Go and related software for your operating system:
  - **MacOS:** Install Homebrew and GoLang. See [Step 1.2](#).
  - **Ubuntu:** Install GoLang. See [Step 1.2](#).
  - **Windows:** Install GoLang. See [Step 1.2](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

## Create Go project and dependencies

1. From the terminal, create a new project folder called **SqlServerSample**.

```
mkdir SqlServerSample
```

2. Change directory to **SqlServerSample** and get 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. Using your favorite text editor, create a file called **CreateTestData.sql** in the **SqlServerSample** folder. Copy and paste the following the T-SQL code inside it. This code 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. Connect to the database using sqlcmd and run the SQL script to create the schema, table, and insert some rows. 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. Open the file and replace its contents with the following code. Add the appropriate values for your server, database, username, and password. This example uses the GoLang Context methods to ensure that there is an active connection to the database server.

```
package main

import (
    _ "github.com/denisenkom/go-mssql"
    "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.Printf("Deleted %d row(s) successfully.\n", deletedRows)
}

// 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()

```

```

ctx := context.Background()

// Check if database is alive.
err := db.PingContext(ctx)
if err != nil {
    return -1, err
}

tsql := fmt.Sprintf("UPDATE TestSchema.Employees SET Location = @Location WHERE Name = @Name")

// Execute non-query with named parameters
result, err := db.ExecContext(
    ctx,
    tsql,
    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
    }

    tsql := fmt.Sprintf("DELETE FROM TestSchema.Employees WHERE Name = @Name;")

    // Execute non-query with named parameters
    result, err := db.ExecContext(ctx, tsql, sql.Named("Name", name))
    if err != nil {
        return -1, err
    }

    return result.RowsAffected()
}

```

## Run the code

- At the command prompt, run the following commands:

```
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](#)
- [Go Driver for Microsoft SQL Server](#)
- [Report issues or ask questions](#)

# Use Java to query an Azure SQL database

9/24/2018 • 2 minutes to read • [Edit Online](#)

This quickstart demonstrates how to use [Java](#) to connect to an Azure SQL database and then use Transact-SQL statements to query data.

## Prerequisites

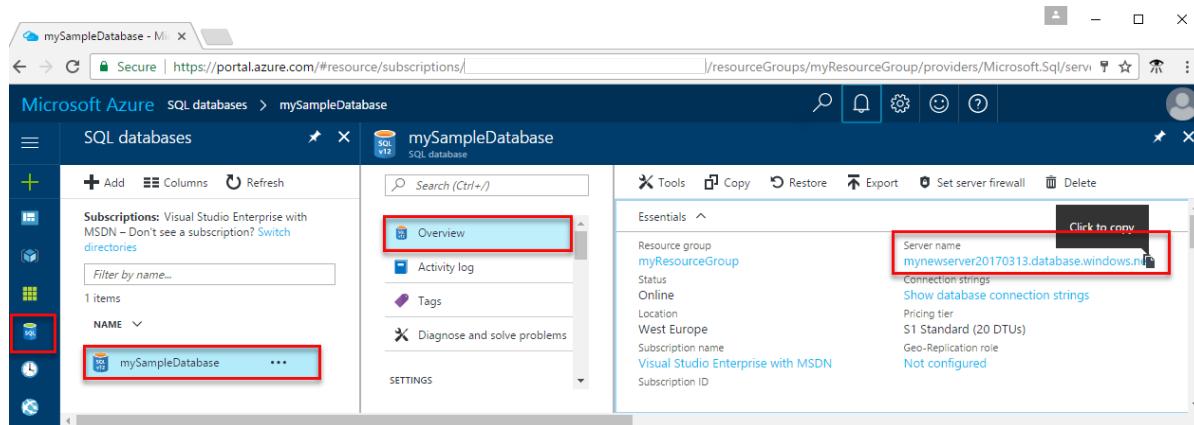
To complete this quickstart, make sure you have the following prerequisites:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed Java and related software for your operating system:
  - **MacOS:** Install Homebrew and Java, and then install Maven. See [Step 1.2 and 1.3](#).
  - **Ubuntu:** Install the Java Development Kit, and install Maven. See [Step 1.2, 1.3, and 1.4](#).
  - **Windows:** Install the Java Development Kit, and Maven. See [Step 1.2 and 1.3](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

## Create Maven project and dependencies

1. From the terminal, create a new Maven project called **sqltest**.

```
mvn archetype:generate "-DgroupId=com.sqlsamples" "-DartifactId=sqltest" "-DarchetypeArtifactId=maven-archetype-quickstart" "-Dversion=1.0.0"
```

2. Enter **Y** when prompted.
3. Change directory 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>6.4.0.jre8</version>
</dependency>
```

4. 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>
```

5. Save and close **pom.xml**.

## Insert code to query SQL database

1. You should already have a file called **App.java** in your Maven project located at:  
..\\sqltest\\src\\main\\java\\com\\sqlsamples\\App.java
2. Open the file and replace its contents with the following code and 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";
        String dbName = "your_database";
        String user = "your_username";
        String password = "your_password";
        String url =
String.format("jdbc:sqlserver://%" + hostName + ";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();
            }
        }
    }
}

```

## Run the code

- At the command prompt, run the following commands:

```

mvn package
mvn -q exec:java "-Dexec.mainClass=com.sql dbsamples.App"

```

- Verify that the top 20 rows are returned and then close the application window.

## Next steps

- [Design your first Azure SQL database](#)
- [Microsoft JDBC Driver for SQL Server](#)
- [Report issues/ask questions](#)

# Use Node.js to query an Azure SQL database

9/24/2018 • 3 minutes to read • [Edit Online](#)

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.

## Prerequisites

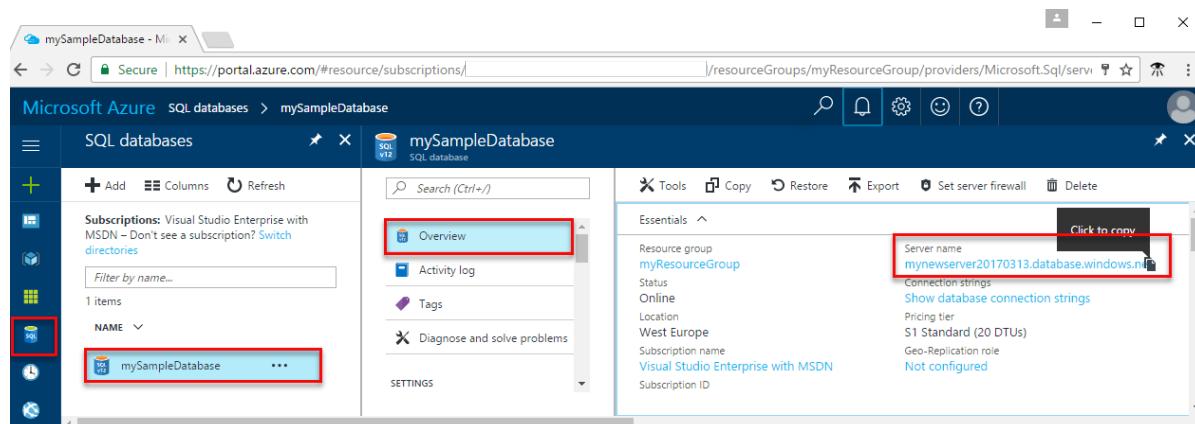
To complete this quickstart, make sure you have the following:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed Node.js and related software for your operating system:
  - **MacOS:** Install Homebrew and Node.js, and then install the ODBC driver and SQLCMD. See [Step 1.2 and 1.3](#).
  - **Ubuntu:** Install Node.js, and then install the ODBC driver and SQLCMD. See [Step 1.2 and 1.3](#).
  - **Windows:** Install Chocolatey and Node.js, and then install the ODBC driver and SQL CMD. See [Step 1.2 and 1.3](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

#### **IMPORTANT**

You must have a firewall rule in place for the public IP address of the computer on which you perform this tutorial. If you are on a different computer or have a different public IP address, create a [server-level firewall rule using the Azure portal](#).

## Create a Node.js project

Open a command prompt and create a folder named *sqltest*. Navigate to the folder you created and run the following command:

```
npm init -y  
npm install tedious  
npm install async
```

## Insert code to query SQL database

1. In your development environment or favorite text editor, create a new file, **sqltest.js**.
2. Replace the contents with the following code and add the appropriate values for your server, database, user, and password.

```

var Connection = require('tedious').Connection;
var Request = require('tedious').Request;

// Create connection to database
var config =
{
    userName: 'someuser', // update me
    password: 'somepassword', // update me
    server: 'edmacasqlserver.database.windows.net', // update me
    options:
    {
        database: 'somedb' //update me
        , encrypt: true
    }
}
var connection = new Connection(config);

// Attempt to connect and execute queries if connection goes through
connection.on('connect', function(err)
{
    if (err)
    {
        console.log(err)
    }
    else
    {
        queryDatabase()
    }
});
);

function queryDatabase()
{ console.log('Reading rows from the Table...');

    // Read all rows from table
    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",
        function(err, rowCount, rows)
        {
            console.log(rowCount + ' row(s) returned');
            process.exit();
        }
    );

    request.on('row', function(columns) {
        columns.forEach(function(column) {
            console.log("%s\t%s", column.metadata.colName, column.value);
        });
    });
    connection.execSql(request);
}
}

```

## Run the code

- At the command prompt, run the following commands:

```
node sqltest.js
```

- Verify that the top 20 rows are returned and then close the application window.

## Next steps

- Learn about the [Microsoft Node.js Driver for SQL Server](#)
- 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](#).
- Learn how to [Connect and query with SSMS](#)
- Learn how to [Connect and query with Visual Studio Code](#).

# Use PHP to query an Azure SQL database

9/24/2018 • 2 minutes to read • [Edit Online](#)

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.

## Prerequisites

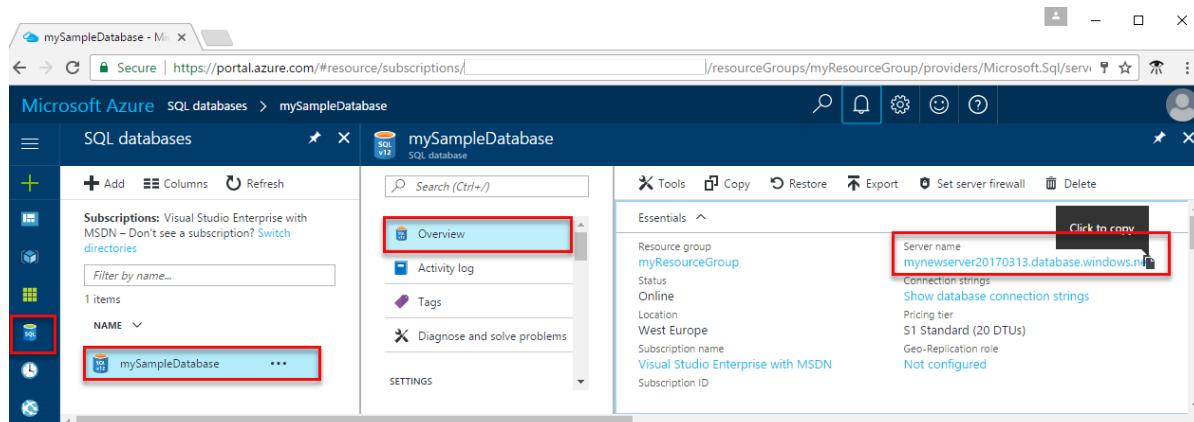
To complete this quickstart, make sure you have the following:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed PHP and related software for your operating system:
  - **MacOS:** Install Homebrew and PHP, install the ODBC driver and SQLCMD, and then install the PHP Driver for SQL Server. See [Steps 1.2, 1.3, and 2.1](#).
  - **Ubuntu:** Install PHP and other required packages, and then install the PHP Driver for SQL Server. See [Steps 1.2 and 2.1](#).
  - **Windows:** Install the newest version of PHP for IIS Express, the newest version of Microsoft Drivers for SQL Server in IIS Express, Chocolatey, the ODBC driver, and SQLCMD. See [Steps 1.2 and 1.3](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

## Insert code to query SQL database

1. In your favorite text editor, create a new file, **sqltest.php**.
2. Replace the contents with the following code and add the appropriate values for your server, database, user, and password.

```
<?php
$serverName = "your_server.database.windows.net";
$connectionOptions = array(
    "Database" => "your_database",
    "Uid" => "your_username",
    "PWD" => "your_password"
);
//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 following commands:

```
php sqltest.php
```

2. Verify that the top 20 rows are returned and then close the application 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](#)

# Use Python to query an Azure SQL database

9/24/2018 • 2 minutes to read • [Edit Online](#)

This quickstart demonstrates how to use [Python](#) to connect to an Azure SQL database and use Transact-SQL statements to query data. For further sdk details, checkout our [reference](#) documentation, a pyodbc [sample](#), and the [pyodbc](#) GitHub repository.

## Prerequisites

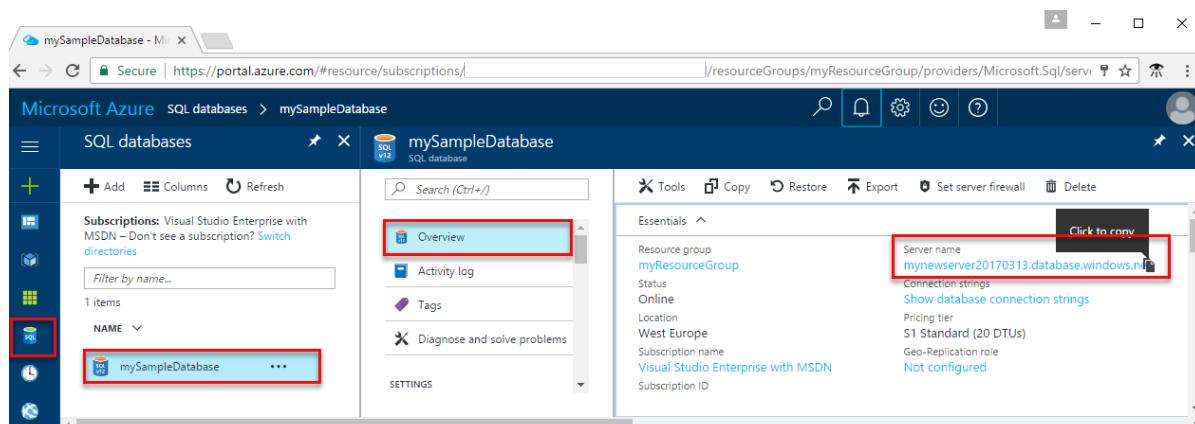
To complete this quickstart, make sure you have the following:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed Python and related software for your operating system:
  - **MacOS:** Install Homebrew and Python, install the ODBC driver and SQLCMD, and then install the Python Driver for SQL Server. See [Steps 1.2, 1.3, and 2.1](#).
  - **Ubuntu:** Install Python and other required packages, and then install the Python Driver for SQL Server. See [Steps 1.2, 1.3, and 2.1](#).
  - **Windows:** Install the newest version of Python (environment variable is now configured for you), install the ODBC driver and SQLCMD, and then install the Python Driver for SQL Server. See [Step 1.2, 1.3, and 2.1](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server

admin name. If necessary, reset the password.

## Insert code to query SQL database

1. In your favorite text editor, create a new file, **sqltest.py**.
2. Replace the contents with the following code and add the appropriate values for your server, database, user, and password.

```
import pyodbc
server = 'your_server.database.windows.net'
database = 'your_database'
username = 'your_username'
password = 'your_password'
driver= '{ODBC Driver 13 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 the command prompt, run the following commands:

```
python sqltest.py
```

2. Verify that the top 20 rows are returned and then close the application window.

## Next steps

- [Design your first Azure SQL database](#)
- [Microsoft Python Drivers for SQL Server](#)
- [Python Developer Center](#)

# Use Ruby to query an Azure SQL database

9/24/2018 • 2 minutes to read • [Edit Online](#)

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.

## Prerequisites

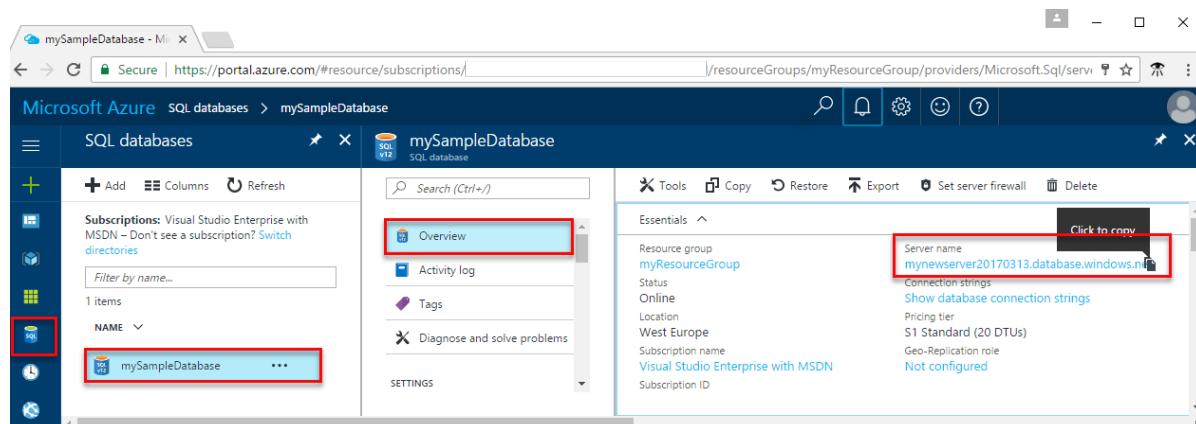
To complete this quickstart, make sure you have the following prerequisites:

- An Azure SQL database. You can use one of these techniques to create a database:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- A [server-level firewall rule](#) for the public IP address of the computer you use for this quickstart.
- You have installed Ruby and related software for your operating system:
  - **MacOS:** Install Homebrew, install rbenv and ruby-build, install Ruby, and then install FreeTDS. See [Step 1.2, 1.3, 1.4, and 1.5](#).
  - **Ubuntu:** Install prerequisites for Ruby, install rbenv and ruby-build, install Ruby, and then install FreeTDS. See [Step 1.2, 1.3, 1.4, and 1.5](#).

## SQL server connection information

Get the connection information needed to connect to the Azure SQL database. You will need the fully qualified server name, database name, and login information in the next procedures.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
3. On the **Overview** page for your database, review the fully qualified server name as shown in the following image. You can hover over the server name to bring up the **Click to copy** option.



4. If you forget your server login information, navigate to the SQL Database server page to view the server admin name. If necessary, reset the password.

## IMPORTANT

You must have a firewall rule in place for the public IP address of the computer on which you perform this tutorial. If you are on a different computer or have a different public IP address, create a [server-level firewall rule using the Azure portal](#).

## Insert code to query SQL database

1. In your favorite text editor, create a new file, **sqltest.rb**
2. Replace the contents with the following code and add the appropriate values for your server, database, user, and password.

```
require 'tiny_tds'
server = 'your_server.database.windows.net'
database = 'your_database'
username = 'your_username'
password = 'your_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 the command prompt, run the following commands:

```
ruby sqltest.rb
```

2. Verify that the top 20 rows are returned and then close the application window.

## Next Steps

- [Design your first Azure SQL database](#)
- [GitHub repository for TinyTDS](#)
- [Report issues or ask questions about TinyTDS](#)
- [Ruby Drivers for SQL Server](#)

# Tutorial: Design your first Azure SQL database using SSMS

9/25/2018 • 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 [SQL Server Management Studio \(SSMS\)](#) to:

- Create a database in the Azure portal\*
- Set up a server-level firewall rule in the Azure portal
- Connect to the database with SSMS
- Create tables with SSMS
- Bulk load data with BCP
- Query that data with SSMS

If you don't have an Azure subscription, [create a free account](#) before you begin.

## NOTE

For the purpose of this tutorial, we are using the [DTU-based purchasing model](#), but you do have the option of choosing the [vCore-based purchasing model](#).

## Prerequisites

To complete this tutorial, make sure you have installed:

- The newest version of [SQL Server Management Studio \(SSMS\)](#).
- The newest version of [BCP](#) and [SQLCMD](#).

## Log in to the Azure portal

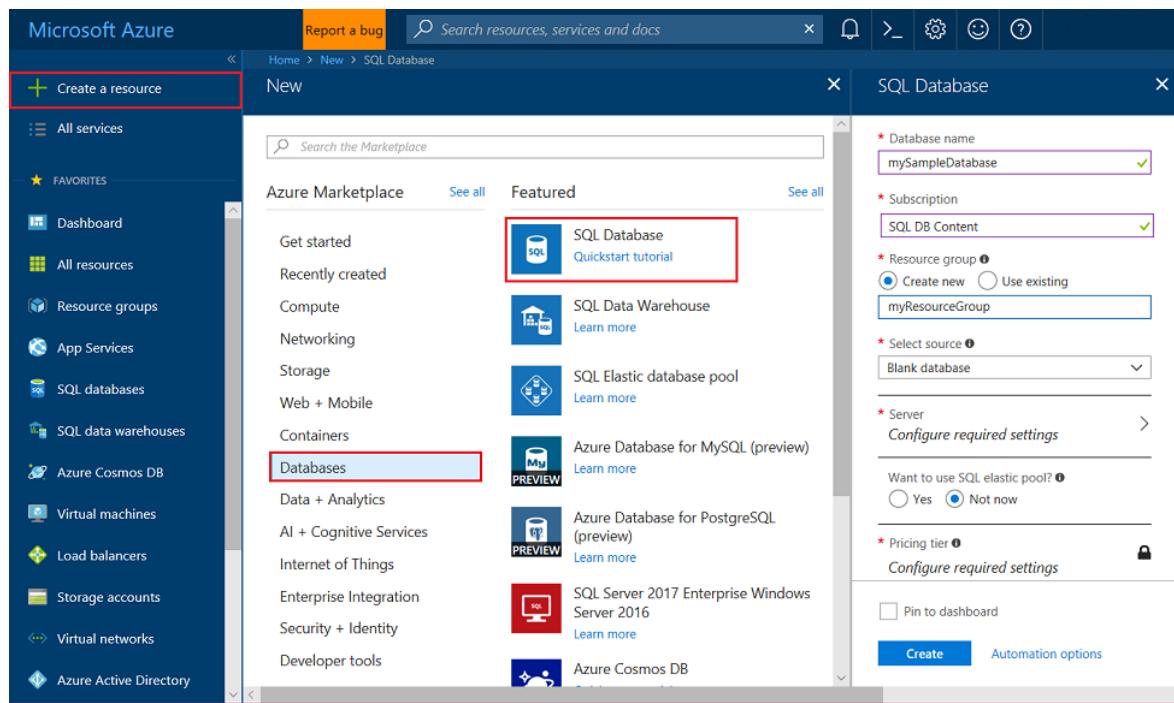
Sign in to the [Azure portal](#).

## Create a blank SQL database

An Azure SQL database is created with a defined set of [compute and storage resources](#). The database is created within an [Azure resource group](#) and in an [Azure SQL Database logical server](#).

Follow these steps to create a blank SQL 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.



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>  | mySampleDatabase  | For valid database names, see <a href="#">Database Identifiers</a> .                |
| <b>Subscription</b>   | Your subscription | For details about your subscriptions, see <a href="#">Subscriptions</a> .           |
| <b>Resource group</b> | myResourceGroup   | 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 create and configure a new server for your new database. 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 contain characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters. |

| SETTING         | SUGGESTED VALUE    | DESCRIPTION  |
|-----------------|--------------------|--|
| <b>Location</b> | Any valid location | For information about regions, see <a href="#">Azure Regions</a> . |

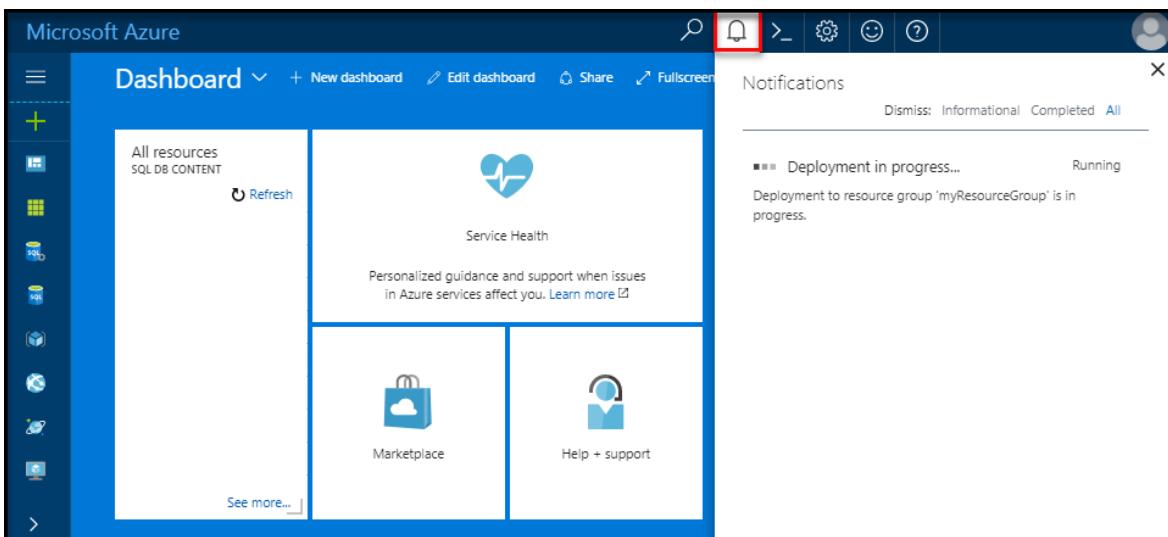
5. Click **Select**.
6. Click **Pricing tier** to specify the service tier, the number of DTUs or vCores, and the amount of storage. Explore the options for the number of DTUs/vCores and storage that is available to you for each service tier. For the purpose of this tutorial, we are using the [DTU-based purchasing model](#), but you do have the option of choosing the [vCore-based purchasing model](#).
7. For this tutorial, select the **Standard** service tier and then use the slider to select **100 DTUs (S3)** and **400 GB** of storage.

8. Accept the preview terms to use the **Add-on Storage** option.

## IMPORTANT

More than 1 TB of storage in the Premium tier is currently available in all regions except the following: UK North, West Central US, UK South2, China East, USDoDCentral, Germany Central, USDoDEast, US Gov Southwest, US Gov South Central, Germany Northeast, China North, US Gov East. In other regions, the storage max in the Premium tier is limited to 1 TB. See [P11-P15 Current Limitations](#).

9. After selecting the server tier, the number of DTUs, and the amount of storage, click **Apply**.
10. Select a **collation** for the blank database (for this tutorial, use the default value). For more information about collations, see [Collations](#)
11. Now that you have completed the SQL Database form, click **Create** to provision the database. Provisioning takes a few minutes.
12. On the toolbar, click **Notifications** to monitor the deployment process.



## Create a server-level firewall rule

The SQL Database service creates a firewall at the server-level that prevents external applications and tools from connecting to the server or any databases on the server unless a firewall rule is created to open the firewall for specific IP addresses. Follow these steps to create a [SQL Database server-level firewall rule](#) for your client's IP address and enable external connectivity through the SQL Database firewall for your IP address only.

### NOTE

SQL Database communicates over port 1433. If you are 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 cannot connect to your Azure SQL Database server unless your IT department opens port 1433.

1. After the deployment completes, click **SQL databases** from the left-hand menu and then click **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 for use to connect to your server and its databases in subsequent tutorials and quickstarts.

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 firewall rule. A firewall rule can open port 1433 for a single IP address or a range of IP addresses.
5. Click **Save**. A server-level firewall rule is created for your current IP address opening port 1433 on the logical server.
6. Click **OK** and then close the **Firewall settings** page.

You can now connect to the SQL Database server and its databases using SQL Server Management Studio or another tool of your choice from this IP address using the server admin account created previously.

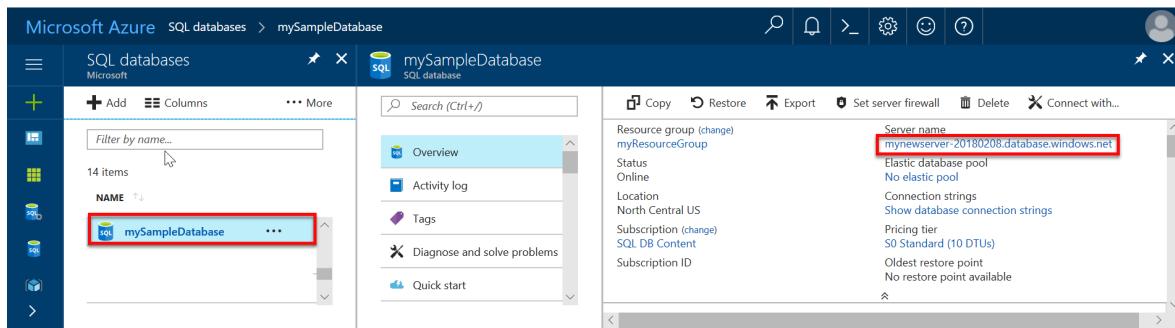
#### IMPORTANT

By default, access through the SQL Database firewall is enabled for all Azure services. Click **OFF** on this page to disable for all Azure services.

## SQL server connection information

Get the fully qualified server name for your Azure SQL Database server in the Azure portal. You use the fully qualified server name to connect to your server using SQL Server Management Studio.

1. Sign in to the [Azure portal](#).
2. Select **SQL Databases** from the left-hand menu and click your database on the **SQL databases** page.
3. In the **Essentials** pane in the Azure portal page for your database, locate and then copy the **Server name**.

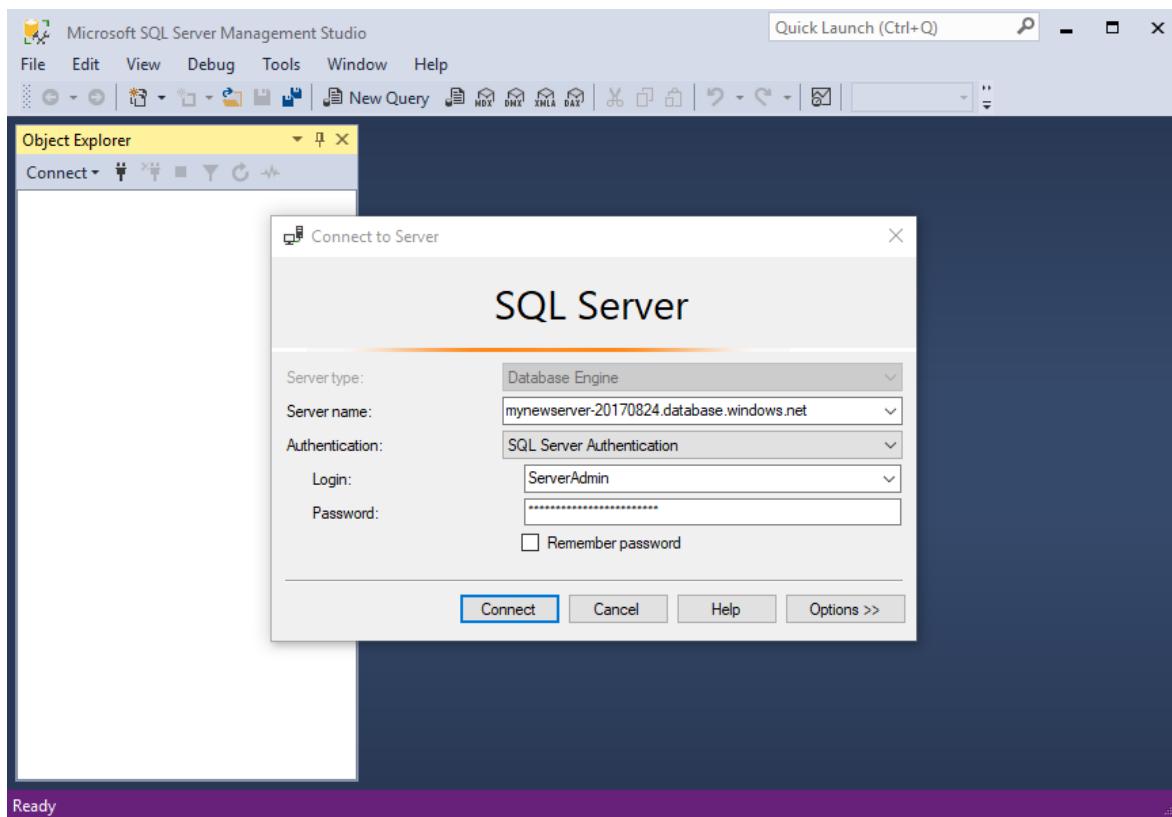


## Connect to the database with SSMS

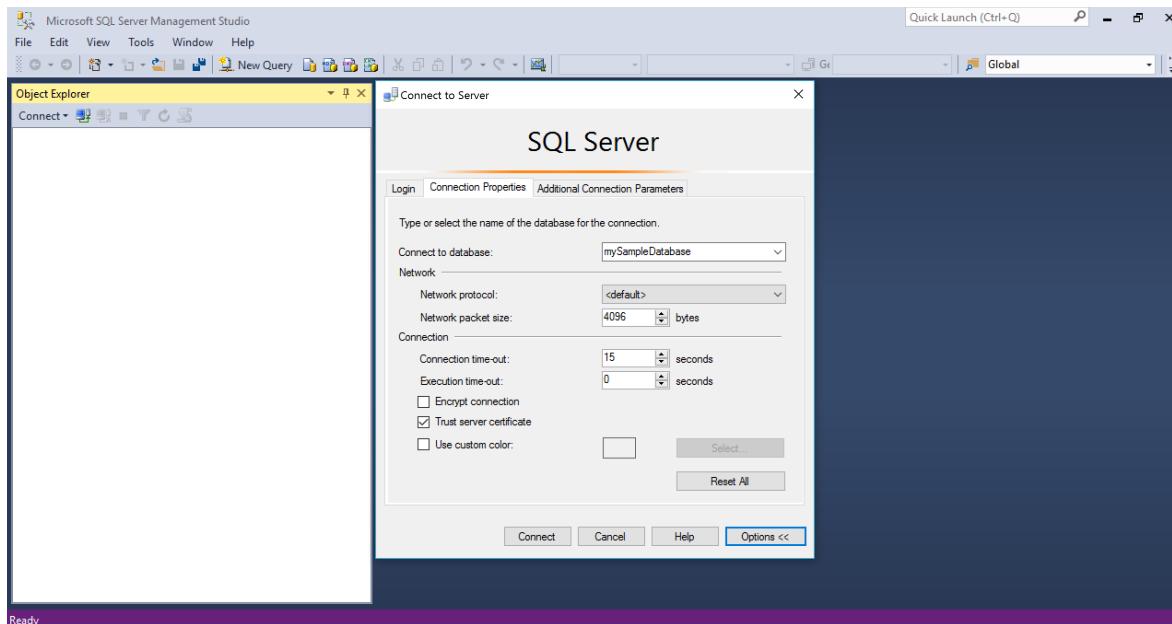
Use [SQL Server Management Studio](#) to establish a connection to your Azure SQL Database server.

1. Open SQL Server Management Studio.
2. In the **Connect to Server** dialog box, enter the following information:

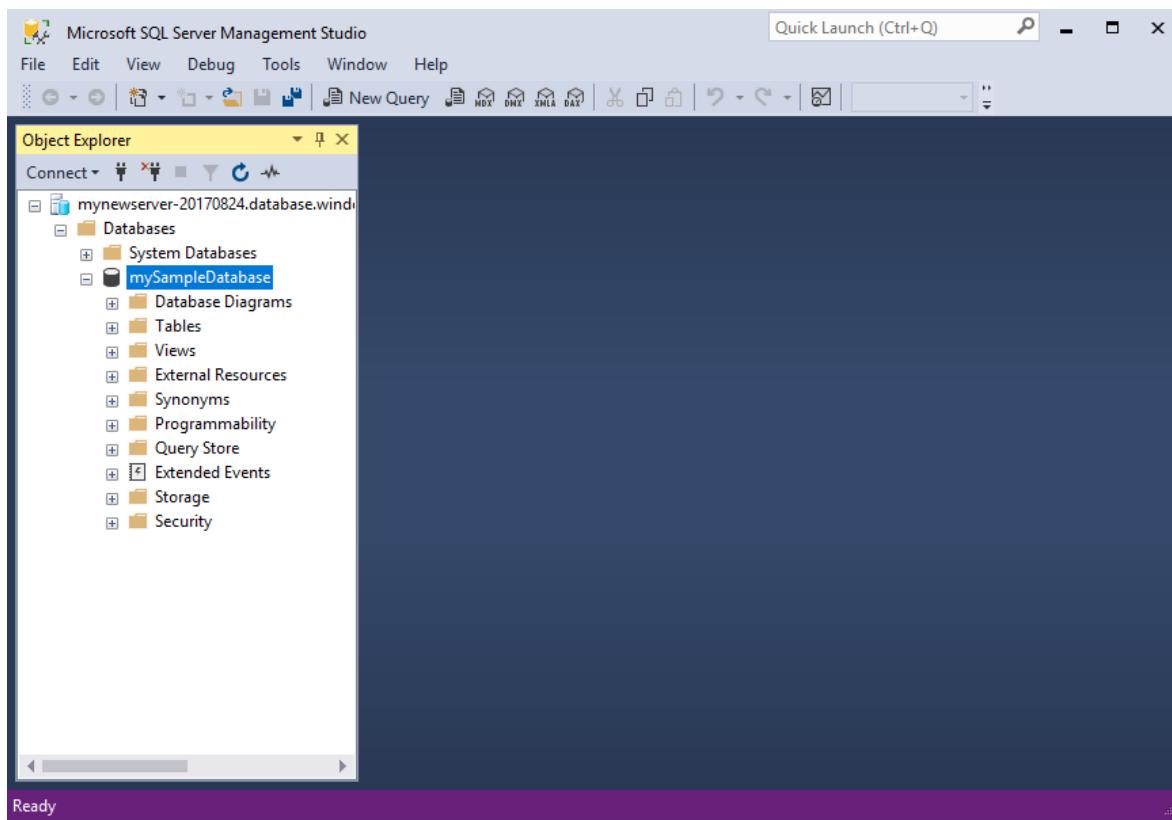
| SETTING        | SUGGESTED VALUE                            | DESCRIPTION  |
|----------------|--|--|
| Server type    | Database engine                            | This value is required   |
| Server name    | The fully qualified server name            | The name should be something like this:<br><b>mynewserver20170824.database.windows.net</b> . |
| Authentication | SQL Server Authentication                  | SQL Authentication is the only authentication type that we have configured in this tutorial. |
| Login          | The server admin account                   | This is the account that you specified when you created the server.                          |
| Password       | The password for your server admin account | This is 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 **mySampleDatabase** to connect to this database.



4. Click **Connect**. The Object Explorer window opens in SSMS.
5. In Object Explorer, expand **Databases** and then expand **mySampleDatabase** to view the objects in the sample database.



## Create tables in the database

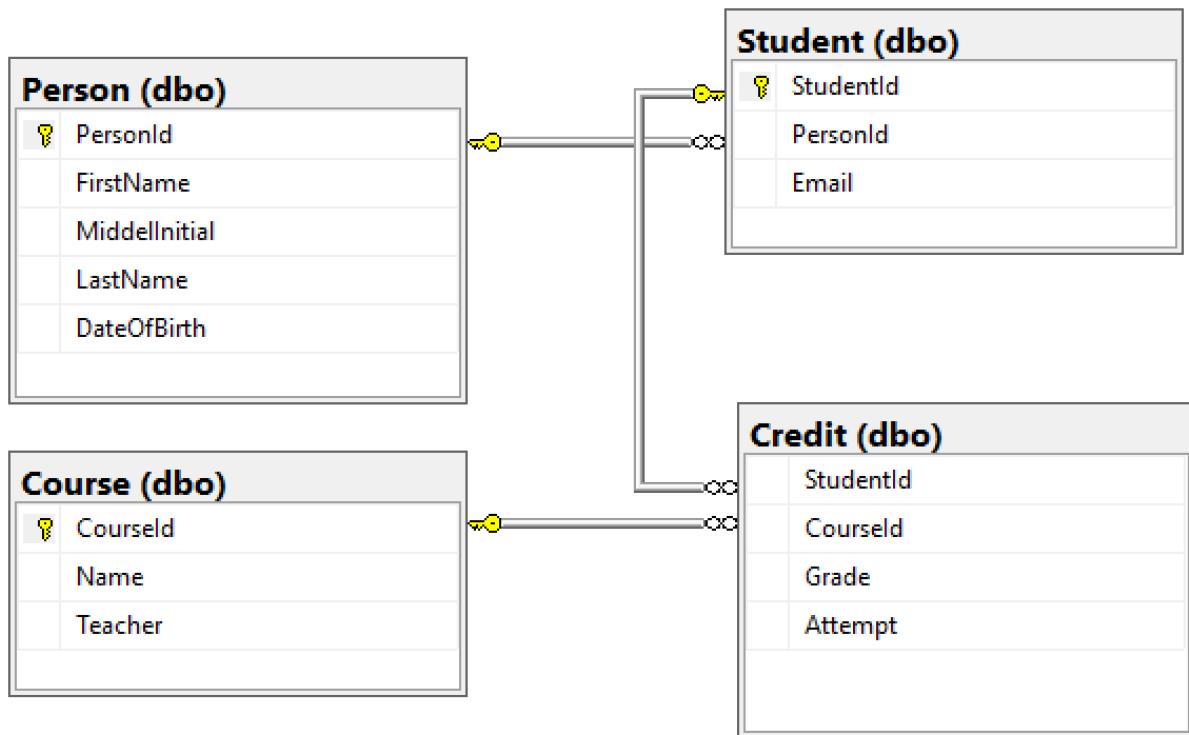
Create a database schema with four tables that model a student management system for universities using [Transact-SQL](#):

- Person
  - Course
  - Student
  - Credit
- that model a student management system for universities

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 **mySampleDatabase** and click **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 SQL Server Management Studio interface. The Object Explorer on the left shows the database structure, including the mySampleDatabase. The central pane displays a query window with the following SQL code:

```
-- Create Course table
CREATE TABLE Course
(
    CourseId INT IDENTITY PRIMARY KEY,
    Name NVARCHAR(50) NOT NULL,
    Teacher NVARCHAR(256) NOT NULL
)

-- Create Credit table
```

The Messages pane at the bottom right shows the command completed successfully.

3. Expand the 'Tables' node in the SQL Server Management Studio Object explorer to see the tables you created.

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left has the 'Tables' node expanded under mySampleDatabase. Four tables are listed: dbo.Course, dbo.Credit, dbo.Person, and dbo.Student. These four tables are highlighted with a red rectangle. The central pane displays the same SQL code as the previous screenshot. The Messages pane at the bottom right shows the command completed successfully.

## Load data into the tables

1. Create a folder called **SampleTableData** in your Downloads folder to store sample data for your database.
2. Right-click the following links and save them into the **SampleTableData** folder.
  - SampleCourseData

- [SamplePersonData](#)
- [SampleStudentData](#)
- [SampleCreditData](#)

3. Open a command prompt window and navigate to the SampleTableData folder.
4. Execute the following commands to insert sample data into the tables replacing the values for **ServerName**, **DatabaseName**, **UserName**, and **Password** with the values for your environment.

```
bcp Course in SampleCourseData -S <ServerName>.database.windows.net -d <DatabaseName> -U <Username> -P <password> -q -c -t ","
bcp Person in SamplePersonData -S <ServerName>.database.windows.net -d <DatabaseName> -U <Username> -P <password> -q -c -t ","
bcp Student in SampleStudentData -S <ServerName>.database.windows.net -d <DatabaseName> -U <Username> -P <password> -q -c -t ","
bcp Credit in SampleCreditData -S <ServerName>.database.windows.net -d <DatabaseName> -U <Username> -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 [Writing SQL Queries](#) to learn more about writing SQL queries. The first query joins all four tables to find all the students taught by 'Dominick Pope' who have a grade higher than 75% in his class. The second query joins all four tables and finds all 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 SQL Server Management Studio query window, execute 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 basic database tasks such as create a database and tables, load and query data, and

restore the database to a previous point in time. You learned how to:

- Create a database
- Set up a 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 an Azure SQL database and connect with C# and ADO.NET](#)

# Design an Azure SQL database and connect with C# and ADO.NET

10/19/2018 • 10 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 database in the Azure portal
- Set up a server-level firewall rule in 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.

## Prerequisites

An installation of [Visual Studio Community 2017](#), [Visual Studio Professional 2017](#), or [Visual Studio Enterprise 2017](#).

## Sign in to the Azure portal

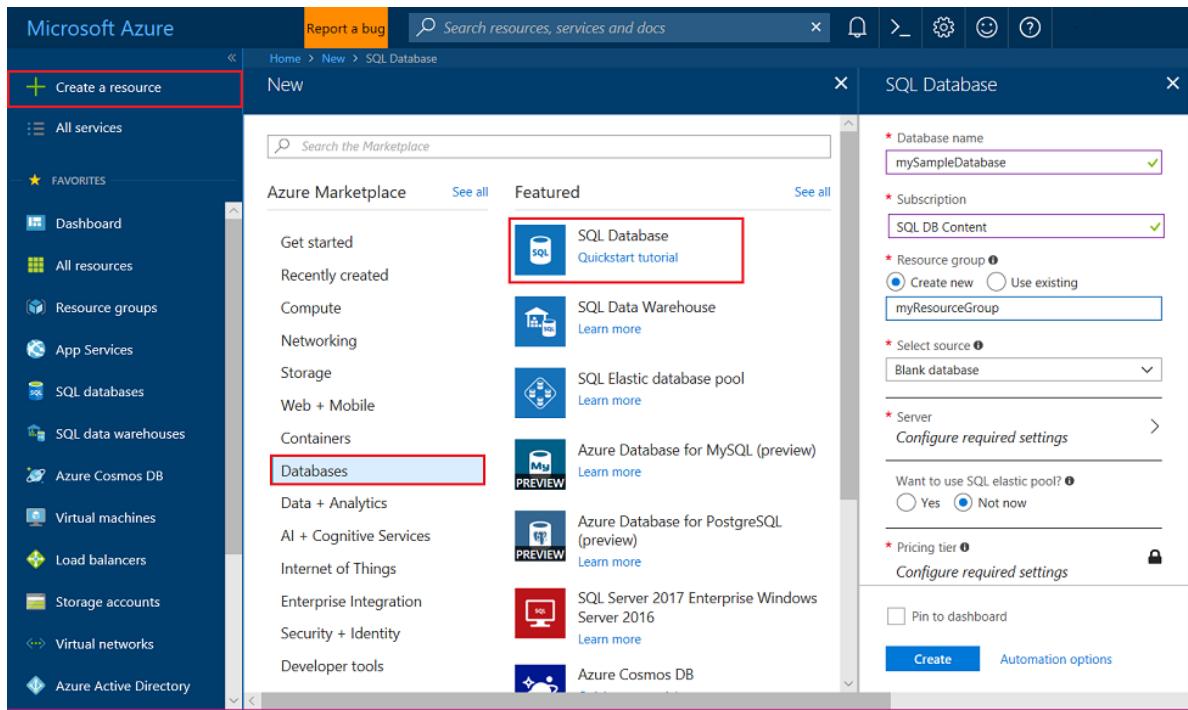
Sign in to the [Azure portal](#).

## Create a blank SQL database

An Azure SQL database is created with a defined set of [compute and storage resources](#). The database is created within an [Azure resource group](#) and in an [Azure SQL Database logical server](#).

Follow these steps to create a blank SQL database.

1. Click **Create a resource** in the upper left-hand corner of the Azure portal.
2. Select **Databases** from the **New** page, and select **Create** under **SQL Database** on the **New** page.



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>  | mySampleDatabase  | For valid database names, see <a href="#">Database Identifiers</a> .                |
| <b>Subscription</b>   | Your subscription | For details about your subscriptions, see <a href="#">Subscriptions</a> .           |
| <b>Resource group</b> | myResourceGroup   | 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 create and configure a new server for your new database. 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 8 characters and must contain characters from three of the following categories: upper case characters, lower case characters, numbers, and non-alphanumeric characters. |

| SETTING  | SUGGESTED VALUE    | DESCRIPTION  |
|----------|--------------------|--|
| Location | Any valid location | For information about regions, see <a href="#">Azure Regions</a> . |

The screenshot shows the Microsoft Azure portal interface for creating a new SQL Server. On the left, the 'Configure required settings' step is highlighted with a red box. On the right, the 'Create a new server' dialog is open, also highlighted with a red box. The 'Server name' field is set to 'mynewserver-20170824'. The 'Server admin login' is 'ServerAdmin', and the 'Password' and 'Confirm password' fields are filled with masked text. The 'Location' is set to 'West Central US'. A checkbox for 'Allow azure services to access server' is checked. At the bottom right of the dialog is a red-bordered 'Select' button.

- Click **Select**.
- Click **Pricing tier** to specify the service tier, the number of DTUs, and the amount of storage. Explore the options for the amount of DTUs and storage that is available to you for each service tier.
- For this tutorial, select the **Standard** service tier and then use the slider to select **100 DTUs (S3)** and **400 GB** of storage.

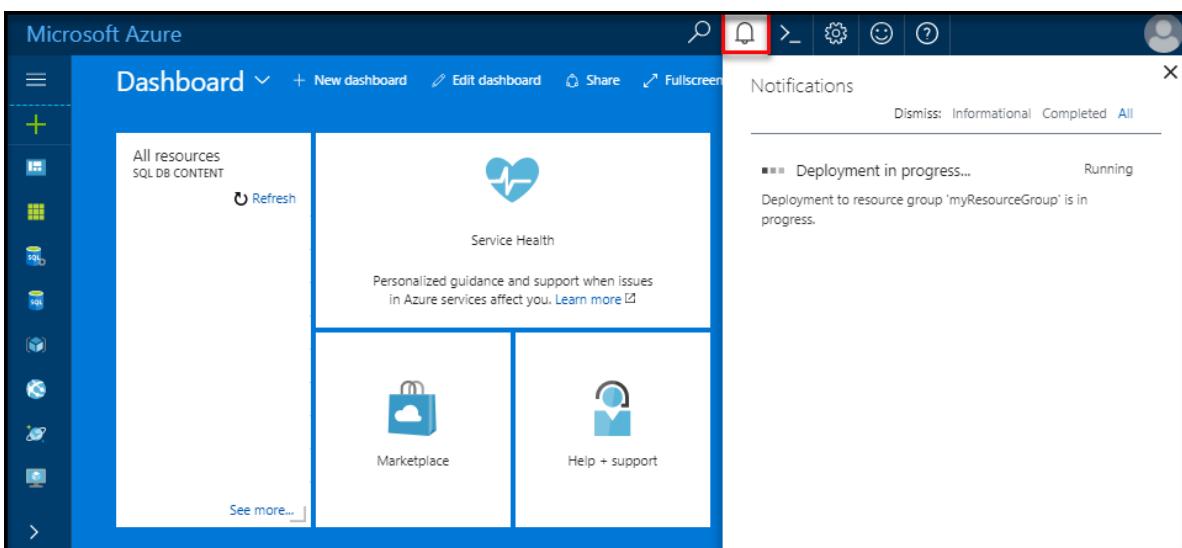
The screenshot shows the 'Configure performance' dialog. The 'Pricing tier' section is highlighted with a red box, showing 'Standard S2: 50 DTU, 250 GB'. The 'Standard' service tier is selected and highlighted with a red box. Below it, the 'DTU (10-3000 DTU) - What is a DTU?' slider is set to 100 (\$3), with a monthly cost of 150.00 USD. The 'Storage (100 MB-1024 GB)' slider is set to 400 GB, with a monthly cost of 12.75 USD. A red box highlights the 'Preview terms' section at the bottom, which says 'Please accept preview terms to use the Add-on Storage option.'

- Accept the preview terms to use the **Add-on Storage** option.

## IMPORTANT

- \* Storage sizes greater than the amount of included storage are in preview and extra costs apply. For details, see [SQL Database pricing](#).
- \* In the Premium tier, more than 1 TB of storage is currently available in the following regions: Canada Central, Canada East, France Central, Germany Central, Japan East, Korea Central, South Central US, South East Asia, US East2, West US, US Gov Virginia, and West Europe. See [P11-P15 Current Limitations](#).

9. After selecting the server tier, the number of DTUs, and the amount of storage, click **Apply**.
10. Select a **collation** for the blank database (for this tutorial, use the default value). For more information about collations, see [Collations](#)
11. Click **Create** to provision the database. Provisioning takes about a minute and a half to complete.
12. On the toolbar, click **Notifications** to monitor the deployment process.



## Create a server-level firewall rule

The SQL Database service creates a firewall at the server-level that prevents external applications and tools from connecting to the server or any databases on the server unless a firewall rule is created to open the firewall for specific IP addresses. Follow these steps to create a [SQL Database server-level firewall rule](#) for your client's IP address and enable external connectivity through the SQL Database firewall for your IP address only.

### NOTE

SQL Database communicates over port 1433. If you are 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 cannot connect to your Azure SQL Database server unless your IT department opens port 1433.

1. After the deployment completes, click **SQL databases** from the left-hand menu and then click **mySampleDatabase** on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified server name (such as **mynewserver20170824.database.windows.net**) and provides options for further configuration.
2. Copy this fully qualified server name for use to connect to your server and its databases in subsequent quick starts.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with icons for storage, databases, and more. The main area is titled 'SQL databases' under 'mySampleDatabase'. A list of 14 items is shown, with 'mySampleDatabase' selected and highlighted with a red box. To the right, there's a detailed view of the database with various settings like resource group, status, location, and subscription information. The 'Server name' field, which contains 'mynewserver-20180208.database.windows.net', is also highlighted with a red box.

- Click **Set server firewall** on the toolbar. The **Firewall settings** page for the SQL Database server opens.

This screenshot shows the 'Firewall settings' page for the logical server 'mynewserver-20180208'. The left sidebar lists resource group, status, location, and subscription details. Below that is a chart showing DTU usage over time. The main panel has a toolbar with 'Set server firewall', 'Save', 'Discard', and 'Add client IP'. The 'Save' and 'Add client IP' buttons are highlighted with red boxes. The 'Add client IP' section shows a table with one rule: 'ClientIPAddress\_2018-2-12' with 'Start IP' 73.42.249.7 and 'End IP' 73.42.249.7. There's also a section for virtual networks.

- Click **Add client IP** on the toolbar to add your current IP address to a new firewall rule. A firewall rule can open port 1433 for a single IP address or a range of IP addresses.
- Click **Save**. A server-level firewall rule is created for your current IP address opening port 1433 on the logical server.
- Click **OK** and then close the **Firewall settings** page.

You can now connect to the SQL Database server and its databases using SQL Server Management Studio or another tool of your choice from this IP address using the server admin account created previously.

#### IMPORTANT

By default, access through the SQL Database firewall is enabled for all Azure services. Click **OFF** on this page to disable for all Azure services.

## SQL server connection information

Get the fully qualified server name for your Azure SQL Database server in the Azure portal. You use the fully qualified server name to connect to your server using SQL Server Management Studio.

- Sign in to the [Azure portal](#).
- Select **SQL Databases** from the left-hand menu, and click your database on the **SQL databases** page.
- In the **Essentials** pane in the Azure portal page for your database, locate and then copy the **Server name**.

The screenshot shows the Microsoft Azure portal interface for managing SQL databases. On the left, there's a sidebar with icons for creating new databases, columns, and more. The main area shows a list of 14 items, with 'mySampleDatabase' selected and highlighted by a red box. To the right, there's a detailed view of the database 'mySampleDatabase'. The 'Overview' section is active. In the top right corner of this view, there's a list of properties: Resource group (myResourceGroup), Status (Online), Location (North Central US), Subscription (change), Pricing tier (S0 Standard (10 DTUs)), and others. The 'Server name' field is explicitly highlighted with a red box, containing the value 'mynewserver-20180208.database.windows.net'.

## C# program example

The next sections of this article present a C# program that uses ADO.NET to send Transact-SQL statements to the SQL database. The C# program performs the following actions:

1. [Connects to our SQL database using ADO.NET.](#)
2. [Creates tables.](#)
3. [Populates the tables with data, by issuing T-SQL INSERT statements.](#)
4. [Updates data by use of a join.](#)
5. [Deletes data by use of a join.](#)
6. [Selects data rows by use of a join.](#)
7. Closes the connection (which drops any temporary tables from tempdb).

The C# program contains:

- C# code to connect to the database.
- Methods that return the T-SQL source code.
- Two methods that submit the T-SQL to the database.

### To compile and run

This C# program is logically one .cs file. But here the program is physically divided into several code blocks, to make each block easier to see and understand. To compile and run this program, do the following:

1. Create a C# project in Visual Studio.
  - The project type should be a *console* application, from something like the following hierarchy:  
**Templates > Visual C# > Windows Classic Desktop > Console App (.NET Framework)**.
2. In the file **Program.cs**, erase the small starter lines of code.
3. Into Program.cs, copy and paste each of the following blocks, in the same sequence they are presented here.
4. In Program.cs, edit the following values in the **Main** method:
  - **cb.DataSource**
  - **cd.UserID**
  - **cb.Password**
  - **InitialCatalog**
5. Verify that the assembly **System.Data.dll** is referenced. To verify, expand the **References** node in the **Solution Explorer** pane.
6. To build the program in Visual Studio, click the **Build** menu.
7. To run the program from Visual Studio, click the **Start** button. The report output is displayed in a cmd.exe window.

## 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 can be useful for demonstration purposes, when no test database is available. Temporary tables are deleted automatically when the connection closes. Any REFERENCES for foreign keys are not enforced for temporary tables.

## C# block 1: Connect by using ADO.NET

- [Next](#)

```
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();
        }
    }
}
```

## C# block 2: T-SQL to create tables

- [Previous](#) / [Next](#)

```

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
);

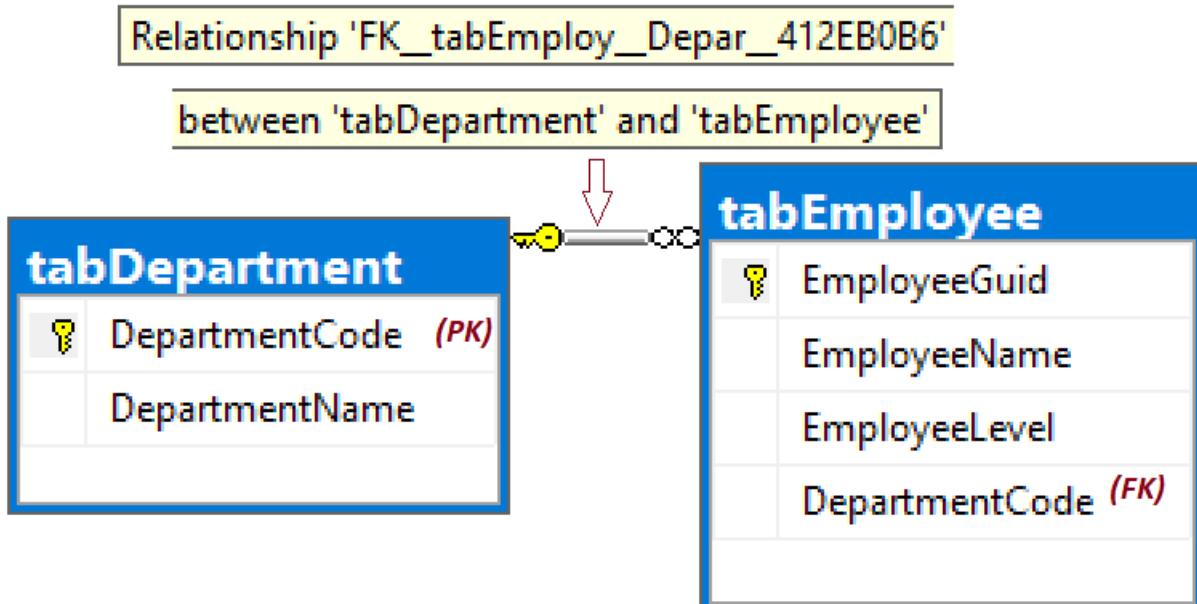
CREATE TABLE tabEmployee
(
    EmployeeGuid uniqueIdentifier not null default NewId()
        PRIMARY KEY,
    EmployeeName nvarchar(128)      not null,
    EmployeeLevel int              not null,
    DepartmentCode nchar(4)          null
        REFERENCES tabDepartment (DepartmentCode) -- (REFERENCES would be disallowed on temporary tables.)
);
";
}

```

#### Entity Relationship Diagram (ERD)

The preceding CREATE TABLE statements involve the **REFERENCES** keyword to create a *foreign key* (FK) relationship between two tables. If you are using tempdb, comment out the `--REFERENCES` keyword using a pair of leading dashes.

Next is an ERD that displays the relationship between the two tables. The values in the `#tabEmployee.DepartmentCode` *child* column are limited to the values present in the `#tabDepartment.Department` *parent* column.



#### C# block 3: T-SQL to insert data

- [Previous](#) / [Next](#)

```

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);
";
}

```

#### C# block 4: T-SQL to update-join

- [Previous](#) / [Next](#)

```

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;
";
}

```

#### C# block 5: T-SQL to delete-join

- [Previous](#) / [Next](#)

```

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;
";
}

```

### C# block 6: T-SQL to select rows

- [Previous](#) / [Next](#)

```

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;
";
}

```

### C# block 6b: ExecuteReader

- [Previous](#) / [Next](#)

This method is designed to run the T-SQL SELECT statement that is built by the **Build\_6\_Tsql\_SelectEmployees** method.

```

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));
            }
        }
    }
}

```

## C# block 7: ExecuteNonQuery

- [Previous](#) / [Next](#)

This method is called for operations that modify the data content of tables without returning any data rows.

```

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.");
    }
}
}

```

## C# block 8: Actual test output to the console

- [Previous](#)

This section captures the output that the program sent to the console. Only the guid values vary between test runs.

```
[C:\csharp_db_test\csharp_db_test\bin\Debug\  
 >> csharp_db_test.exe  
  
=====  
Now, CreateTables (10)...  
  
=====  
Now, Inserts (20)...  
  
=====  
Now, UpdateJoin (30)...  
2 rows affected, by UpdateJoin.  
  
=====  
Now, DeleteJoin (40)...  
  
=====  
Now, SelectEmployees (50)...  
0199be49-a2ed-4e35-94b7-e936acf1cd75 , Alison , 20 , acct , Accounting  
f0d3d147-64cf-4420-b9f9-76e6e0a32567 , Barbara , 17 , hres , Human Resources  
cf4caede-e237-42d2-b61d-72114c7e3afa , Carol , 22 , acct , Accounting  
cdde7727-bcf7-4f72-a665-87199c415f8b , Elle , 15 , NULL , NULL  
  
[C:\csharp_db_test\csharp_db_test\bin\Debug\  
 >>
```

## Next steps

In this tutorial, you learned basic database tasks such as create a database and tables, load and query data, and restore the database to a previous point in time. 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, and delete data
- Query data

Advance to the next tutorial to learn about migrating your data.

[Migrate your SQL Server database to Azure SQL Database](#)

# Migrate SQL Server to Azure SQL Database offline using DMS

10/23/2018 • 9 minutes to read • [Edit Online](#)

You can use the 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 an Azure SQL Database by using the 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 the Azure Database Migration Service.
- Create a migration project by using the 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 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](#) (any edition).
- 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 an instance of Azure SQL Database instance, which you do by following the detail in the article [Create an Azure SQL database in the Azure portal](#).
- Download and install the [Data Migration Assistant](#) v3.3 or later.
- Create a VNET for the 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](#).
- Ensure that your Azure Virtual Network (VNET) Network Security Group rules do not block the following communication ports 443, 53, 9354, 445, 12000. For more detail on Azure VNET 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 the Azure Database Migration Service to access the source SQL Server, which by default is TCP port 1433.

- If you are 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 the 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 the Azure Database Migration Service to access the source database(s) for migration.
- Create a server-level [firewall rule](#) for the Azure SQL Database server to allow the Azure Database Migration Service access to the target databases. Provide the subnet range of the VNET used for the 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 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.

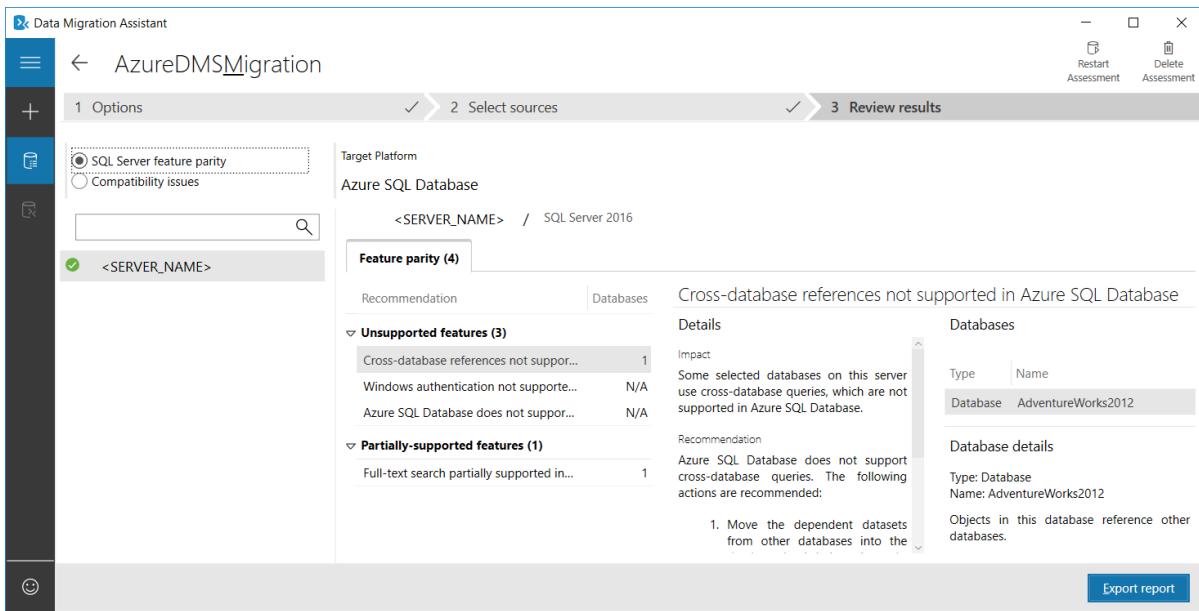
When you are assessing the source SQL Server database migrating to Azure SQL Database, you can choose one or both of the following assessment report types:

- 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**.

When the assessment is complete, the results display as shown in the following graphic:



For Azure SQL Database, the assessments identify feature parity issues and migration blocking issues.

- 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 are comfortable with the assessment and satisfied that the selected database is a viable candidate for migration to Azure SQL Database, use the Data Migration Assistant to migrate the schema to Azure SQL Database.

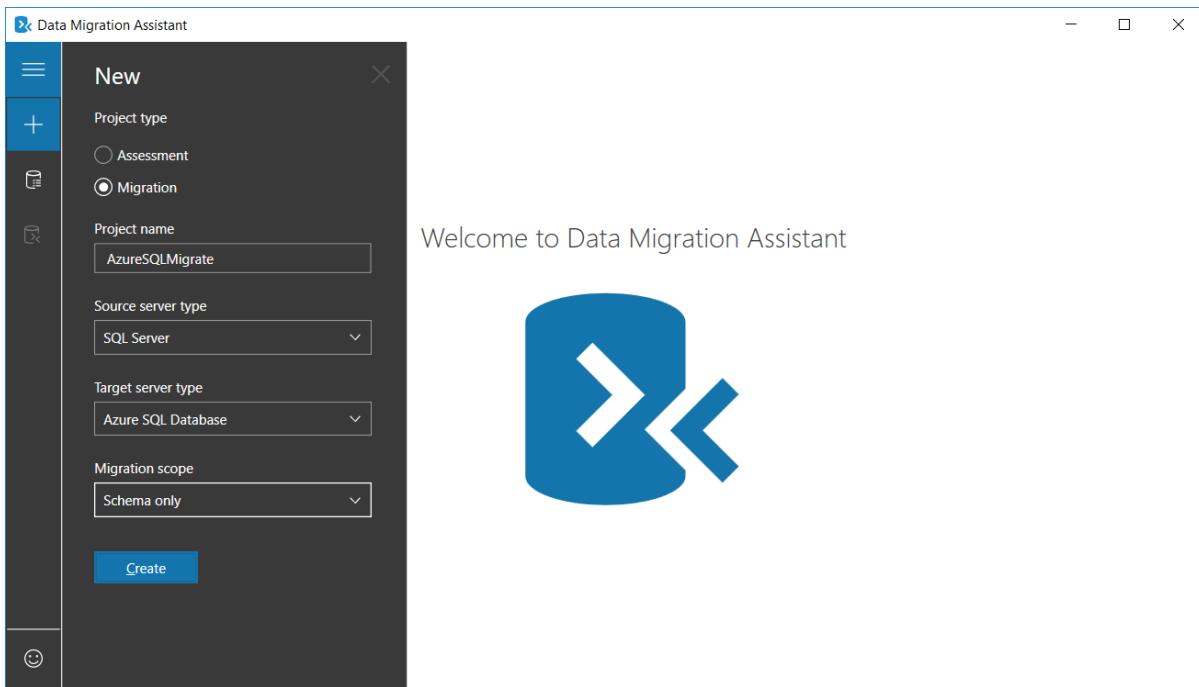
### NOTE

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.

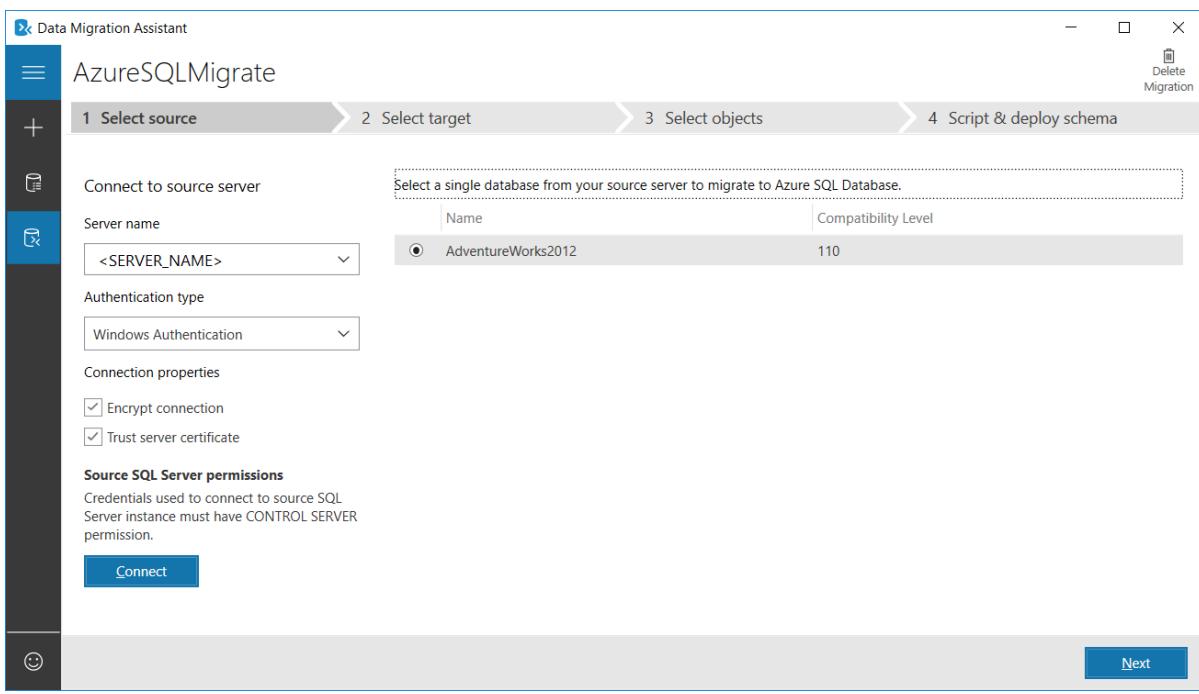
To migrate the **AdventureWorks2012** schema to Azure SQL Database, perform the following steps:

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**.

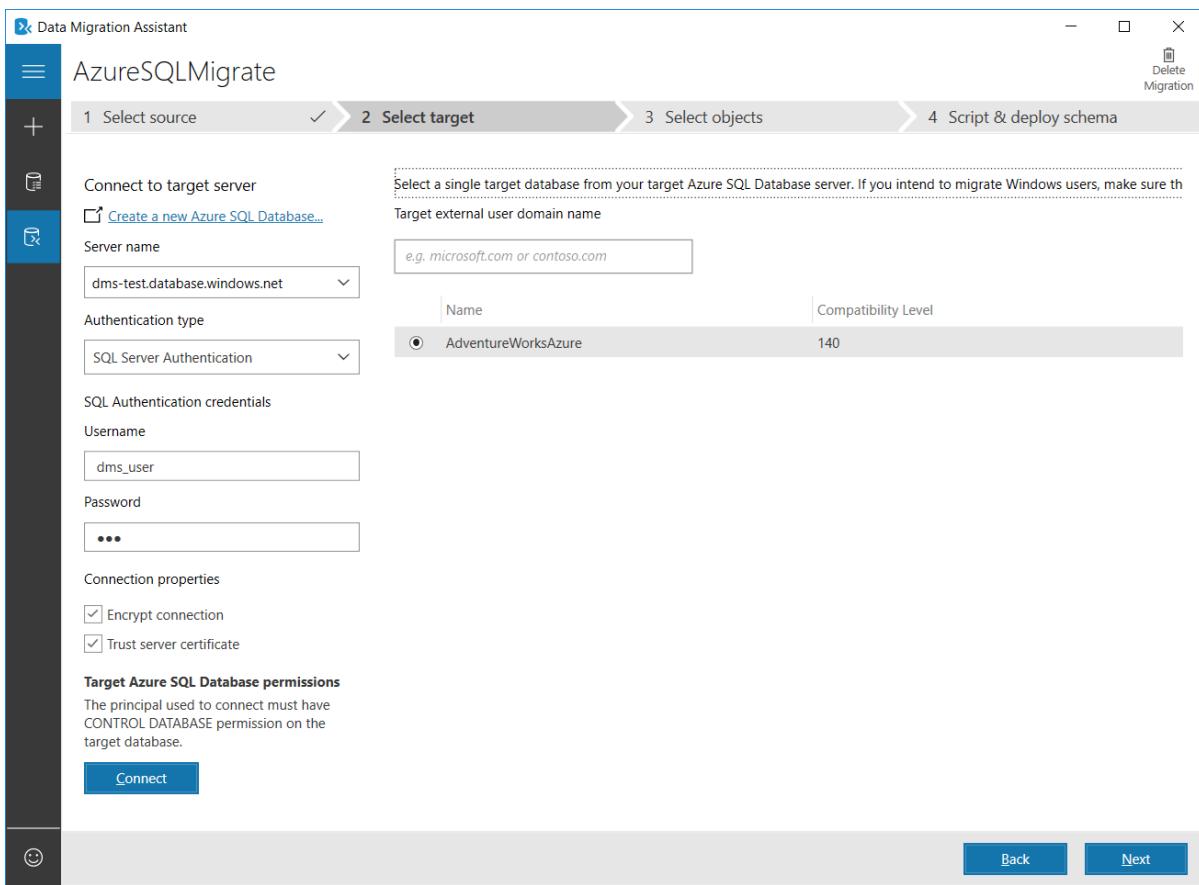
After performing the previous steps, the Data Migration Assistant interface should appear as shown in the following graphic:



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.

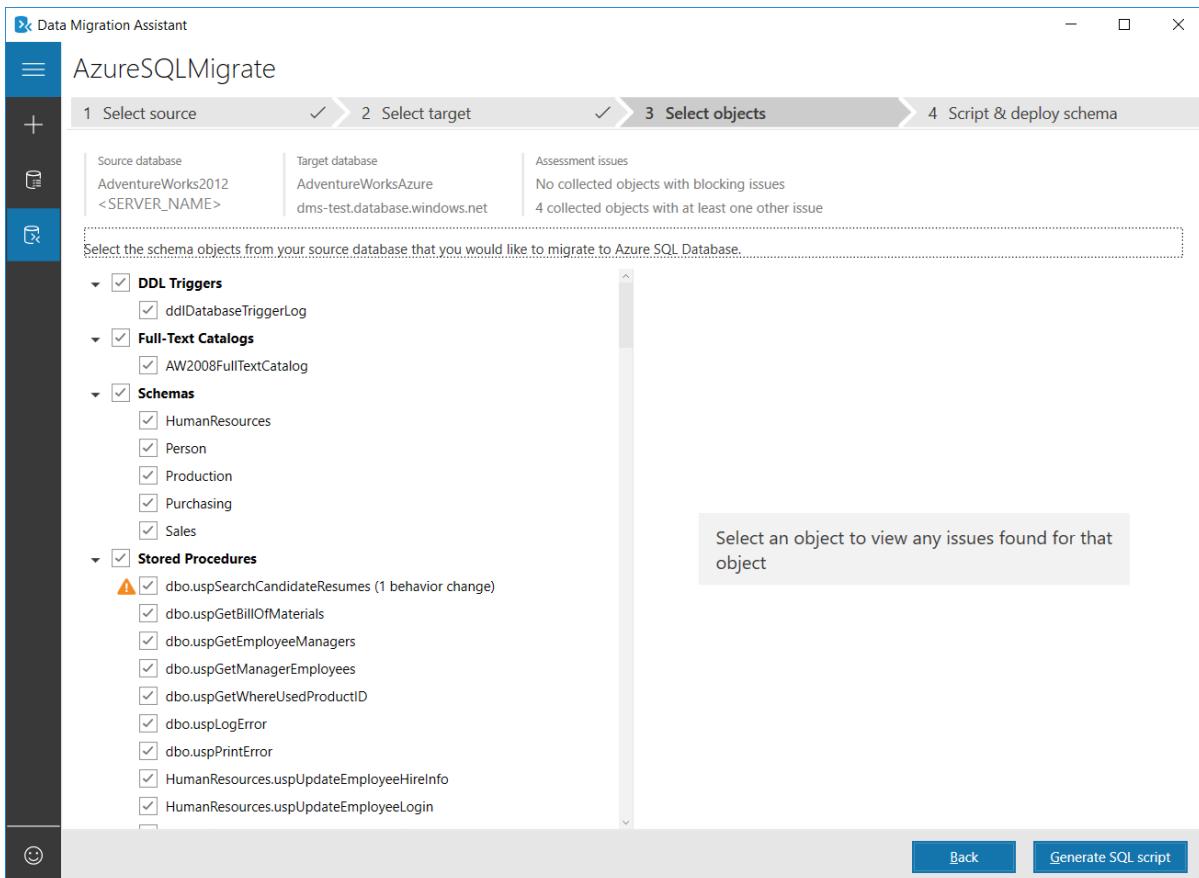


6. Select **Next**, under **Connect to target server**, specify the target connection details for the Azure SQL database, select **Connect**, and then select the **AdventureWorksAzure** database you had pre-provisioned in Azure SQL database.



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.

This screenshot shows the Data Migration Assistant interface for the 'AzureSQLMigrate' project. The current step is '4 Script & deploy schema'. The 'Generated script' pane contains a large block of SQL code intended for deployment. The code includes creating a FullTextCatalog, schemas (HumanResources, Person), and their respective objects like tables and stored procedures. A note at the top of the script pane states: 'This script was generated for the selected schema objects. Review the script, make edits if necessary, and click "Deploy schema" to deploy to Azure SQL Database. Any selected users were not scripted; these will be migrated separately upon clicking "Deploy schema." SQL logins associated with selected users will be recreated with strong, random passwords. You will need to change these passwords and enable them again on the target.' Below the script are 'Back' and 'Deploy schema' buttons.

- 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 Data Migration Assistant interface for the 'AzureSQLMigrate' project, similar to the previous one but with deployment results visible. The 'Deployment results' pane on the right lists 2,236 commands executed, all of which are marked as successful with green checkmarks. The commands include various database operations like creating objects, altering authorizations, and setting ARITHABORT options. The 'Generated script' pane is identical to the previous screenshot. At the bottom, a message says 'Schema migration completed with errors' and provides a duration of '0h 2m 23s'. There are 'Back' and 'Redeploy schema' buttons at the bottom right.

## Register the Microsoft.DataMigration resource provider

- Log in to the Azure portal, select **All services**, and then select **Subscriptions**.

All services Filter By category ▾

GENERAL (14) —

- Dashboard
- Management Groups PREVIEW
- Cost Management + Billing PREVIEW
- Help + support
- Tags
- All resources
- Subscriptions
- Reservations
- Service Health
- What's new

COMPUTE (20) —

- Virtual machines
- Virtual machines (classic)
- Container services
- Function Apps

2. Select the subscription in which you want to create the instance of the Azure Database Migration Service, and then select **Resource providers**.

Home > Subscriptions > <subscription>

## Subscriptions Microsoft

Add

My role: Owner Status: 3 selected

Apply

Search to filter items...

SUBSCRIPTI... ↑↓ SUBSCRIPTION ID ↑↓

<subscription> <subscription ID>

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

Resource providers

Properties

Resource locks

SUPPORT + TROUBLESHOOTING

New support request

3. Search for migration, and then to the right of **Microsoft.DataMigration**, select **Register**.

The screenshot shows the Azure portal interface. On the left, the navigation menu includes 'Create a resource', 'All services', 'Dashboard', 'Resource groups', 'App Services', 'Function Apps', 'SQL databases', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', 'Virtual networks', 'Azure Active Directory', 'Monitor', and 'Advisor'. The main content area is titled '<Subscription>' and shows a list of registered providers. One provider, 'Microsoft.DataMigration', is listed with the status 'NotRegistered'. A red box highlights the 'Register' button next to it.

## Create an 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.

The screenshot shows the Azure portal's 'New' blade. The search bar at the top contains the text 'Azure Database Migration Service', which is highlighted with a red box. Below the search bar, there is a list of service categories: Networking, Storage, Web + Mobile, Containers, Databases, Data + Analytics, AI + Cognitive Services, Internet of Things, Enterprise Integration, Security + Identity, Developer tools, Monitoring + Management, Add-ons, and Blockchain. Under the 'Databases' category, 'Azure Database Migration Service' is listed with a 'PREVIEW' badge and a 'Quickstart tutorial' link. The rest of the blade shows other service options like DevOps Project, Web App, SQL Database, Cosmos DB, Storage Account, and Serverless Function App, along with their respective quickstart tutorials.

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 | <a href="#">Documentation</a><br><a href="#">Privacy Statement</a> |

---

**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 the Azure Database Migration Service.
5. Select an existing virtual network (VNET) or create a new one.

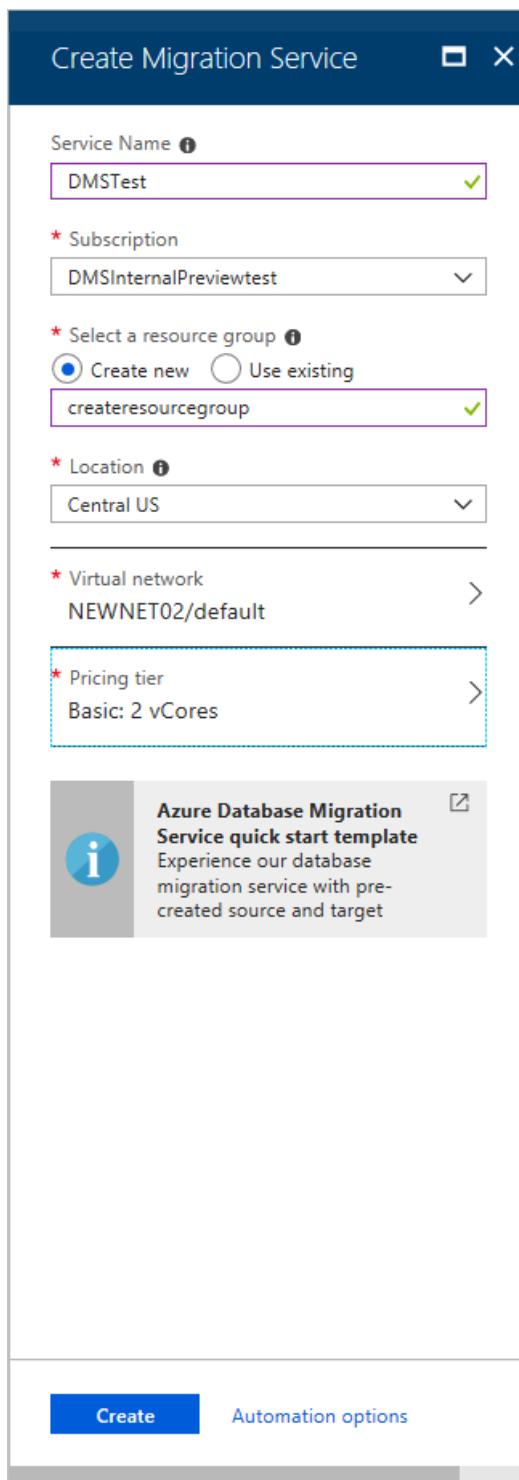
The VNET provides the Azure Database Migration Service with access to the source SQL Server and the target Azure SQL Database instance.

For more information about how to create a VNET 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](#).

If you need help in choosing the right Azure Database Migration Service tier, refer to the recommendations in the posting [here](#).



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.

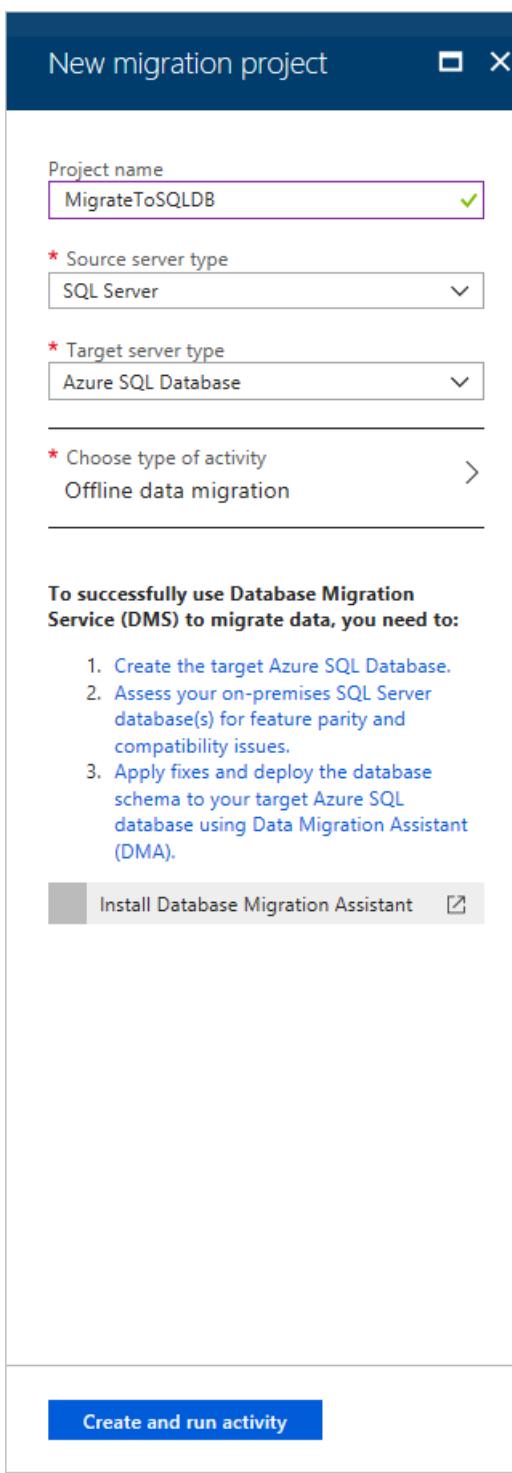
1. In the Azure portal, select **All services**, search for Azure Database Migration Service, and then select **Azure Database Migration Services**.

The screenshot shows the Azure portal's left sidebar with various service icons and names. At the top, there is a search bar with the placeholder "Shift+Space to toggle favorites". Below the search bar, the "Azure Database Migration Services" service is listed under the "FAVORITES" section. The main content area displays the "Azure Database Migration Services" page, which includes a summary card with metrics like "Status: Deploying", "Location: westus2", and "Subscription ID: fc04246f-04c5-437e-ac5e-206a19e7193f". A "Help improve the service menu!" link is located at the bottom right of the page.

2. On the **Azure Database Migration Services** screen, search for the name of the Azure Database Migration Service instance that you created, and then select the instance.

The screenshot shows the "SQL\_Migrate" instance details within the "Azure Database Migration Services" blade. The "Essentials" section displays the instance's configuration, including its Resource group ("SQL\_Migrate"), Network ("AzureDMS-CORP-WUS2-VNET-4825/subnets/Subnet-1"), Subscription name ("DMSInternalPreviewtest"), and Pricing tier ("0"). The "Status" is shown as "Deploying". The "Migration Projects" section is currently empty, displaying the message "No database migration projects to display".

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**, 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

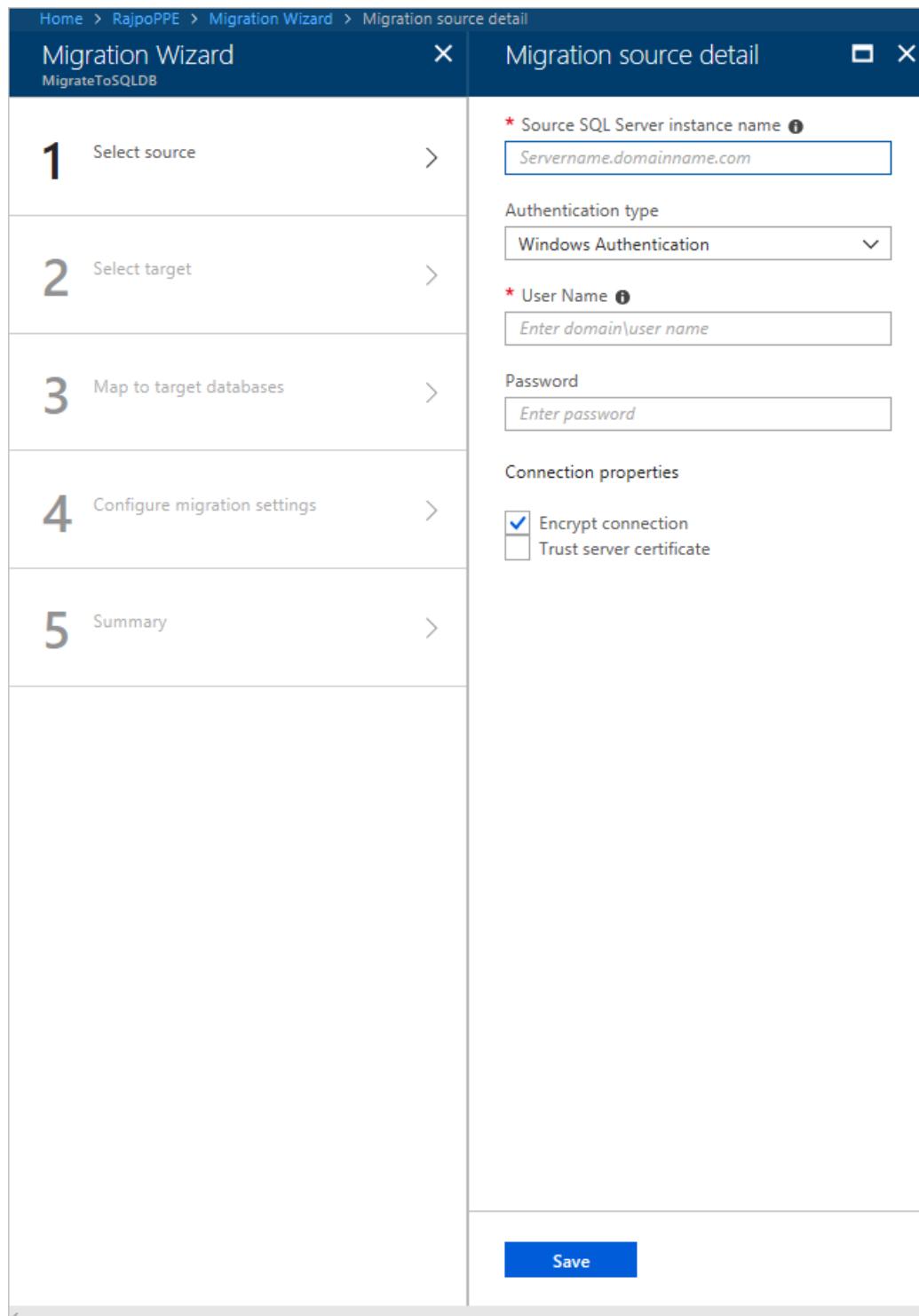
1. On the **Migration source detail** screen, specify the connection details for the source SQL Server instance. Make sure to use a Fully Qualified Domain Name (FQDN) for the source SQL Server instance name. You can also use the IP Address for situations in which DNS name resolution is not possible.
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.

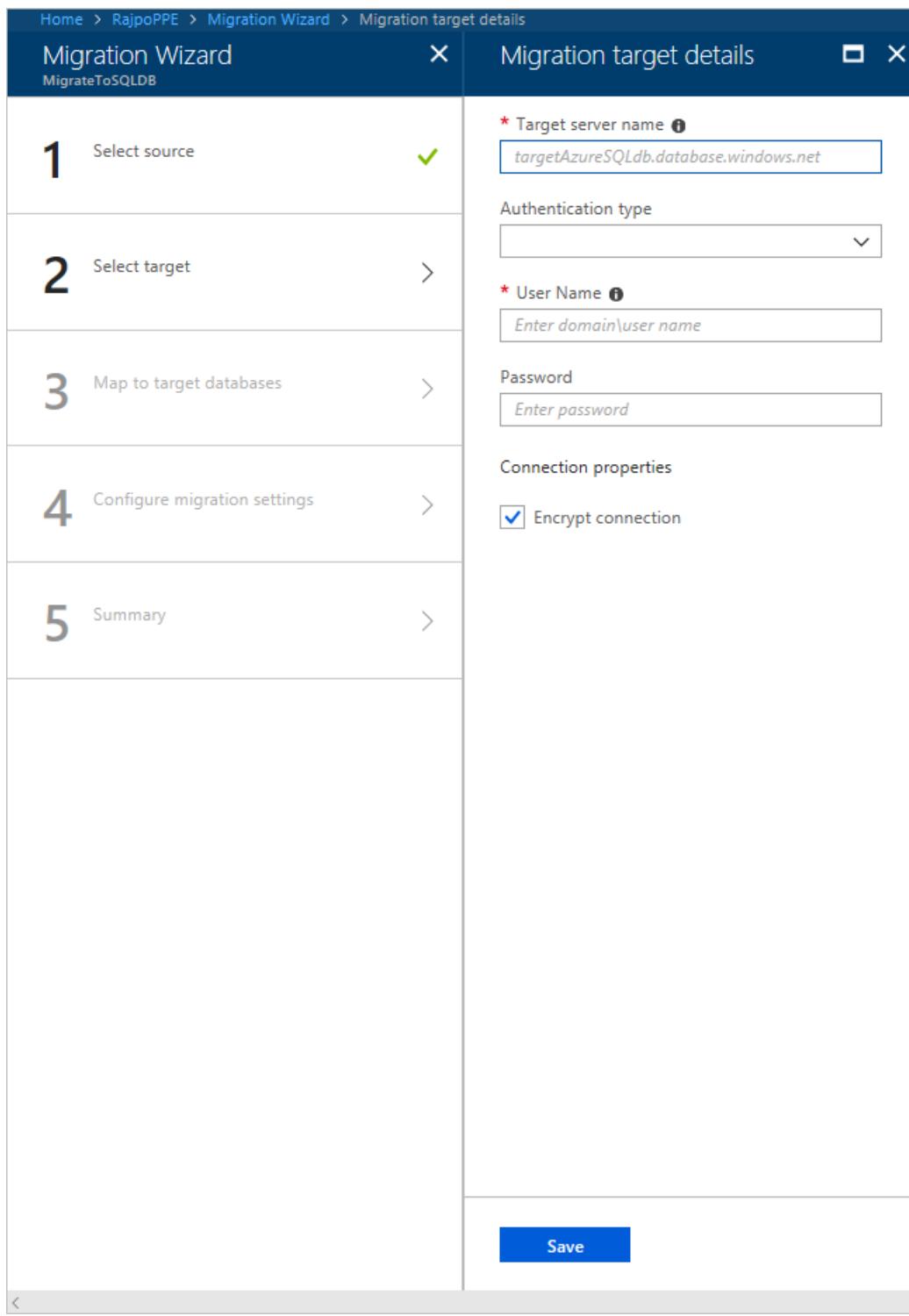
**Caution**

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.



## 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, the Azure Database Migration Service selects the target database by default.

**Migration Wizard** X

MigrateToSQLDB

|          |                              |   |
|----------|------------------------------|---|
| <b>1</b> | Select source                | ✓ |
| <b>2</b> | Select target                | ✓ |
| <b>3</b> | Map to target databases      | > |
| <b>4</b> | Configure migration settings | > |
| <b>5</b> | Summary                      | > |

**Map to target databases** X

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.

Search to filter items... All

6 item(s)

| SOURCE DATABASE  | SIZE      | TARGET DATABASE     | SET SOURCE DB READ-ONLY  |
|--|-----------|---------------------|--------------------------|
| <input checked="" type="checkbox"/> AdventureWorks2012 | 230.06 MB | AdventureWorksAzure | <input type="checkbox"/> |
| <input type="checkbox"/> contosopayrolldb              | 5.00 MB   |                     | <input type="checkbox"/> |
| <input type="checkbox"/> HRMinDowntime                 | 147.31 MB |                     | <input type="checkbox"/> |
| <input type="checkbox"/> SitesEE                       | 372.00 MB |                     | <input type="checkbox"/> |
| <input type="checkbox"/> StackOverflowEE               | 486.00 MB |                     | <input type="checkbox"/> |
| <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.

Note that the 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** X

MigrateToSQLDB

|          |                              |   |
|----------|------------------------------|---|
| <b>1</b> | Select source                | ✓ |
| <b>2</b> | Select target                | ✓ |
| <b>3</b> | Map to target databases      | ✓ |
| <b>4</b> | Configure migration settings | ✓ |
| <b>5</b> | Summary                      | > |

**Select tables** X

Database settings

AdventureWorks2012 71 of 89

Search to filter items... All

71 item(s)

NAME

|   |   |
|---|---|
| <input checked="" type="checkbox"/> Person.Address          | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Person.AddressType      | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> dbo.AWBuildVersion      | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Production.BillOfMat... | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Person.BusinessEntity   | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Person.BusinessEntit... | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Person.BusinessEntit... | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Person.ContactType      | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Person.CountryRegion    | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Sales.CountryRegion...  | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Sales.CreditCard        | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Production.Culture      | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Sales.Currency          | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Sales.CurrencyRate      | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> Sales.Customer          | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> dbo.DatabaseLog         | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> HumanResources.De...    | The target table is not empty. If selected for migration, all the records in the table will be deleted... |
| <input checked="" type="checkbox"/> 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**: A sidebar with steps 1-5. Step 4 is expanded, showing migration details like source and target servers, database names, and activity type.
- Migration summary**: The main summary page with a 'Run migration' button.
- Choose validation option**: A modal window where 'Validate my database(s)' is selected. It also lists validation options: Schema comparison, Data consistency, and Query correctness, all of which are checked.

- Select **Save**, review the summary to ensure that the source and target details match what you previously specified.

The screenshot shows two windows side-by-side: the 'Migration Wizard' and the 'Migration summary'.

**Migration Wizard (Left Window):**

- Step 1:** Select source (Completed)
- Step 2:** Select target (Completed)
- Step 3:** Map to target databases (Completed)
- Step 4:** Configure migration settings (Completed)
- Step 5:** Summary (Incomplete, indicated by a right-pointing arrow)

**Migration summary (Right Window):**

**Information:**

- 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 (Completed)

**Source server name:** 10.105.129.164

**Source server version:** SQL Server 2012  
11.0.7462.6

**Target server name:** dmazuresqlldbserver.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** (Blue button at the bottom)

## 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, 11.0.7462.6), and 'Target version' (Azure SQL Database, 12.0.2000.8). 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 \(DMS\) hands-on lab.](#)

# Secure your Azure SQL Database

10/23/2018 • 10 minutes to read • [Edit Online](#)

SQL Database secures your data by:

- Limiting access to your database using firewall rules
- Using authentication mechanisms that require their identity
- Authorization to data through role-based memberships and permissions,
- Row-level security
- Dynamic data masking

SQL Database also has sophisticated monitoring, auditing, and threat detection.

You can improve the protection of your database against malicious users or unauthorized access with just a few simple steps. In this tutorial you learn to:

- Set up server-level firewall rules for your server in the Azure portal
- Set up database-level firewall rules for your database using SSMS
- Connect to your database using a secure connection string
- Manage user access
- Protect your data with encryption
- Enable SQL Database auditing
- Enable SQL Database threat detection

If you don't have an Azure subscription, [create a free account](#) before you begin.

## Prerequisites

To complete this tutorial, make sure you have the following:

- Installed the newest version of [SQL Server Management Studio \(SSMS\)](#).
- Installed Microsoft Excel
- Created an Azure SQL server and database - See [Create an Azure SQL database in the Azure portal](#), [Create a single Azure SQL database using the Azure CLI](#), and [Create a single Azure SQL database using PowerShell](#).

## Log in to the Azure portal

Sign in to the [Azure portal](#).

## Create a server-level firewall rule in the Azure portal

SQL databases are protected by a firewall in Azure. By default, all connections to the server and the databases inside the server are rejected except for connections from other Azure services. For more information, see [Azure SQL Database server-level and database-level firewall rules](#).

The most secure configuration is to set 'Allow access to Azure services' to OFF. If you need to connect to the database from an Azure VM or cloud service, you should create a [Reserved IP \(classic deployment\)](#) and allow only the reserved IP address access through the firewall. If you are using the [Resource Manager](#) deployment model, a dedicated Public IP address is assigned to the resource, and you should allow this IP address through the firewall.

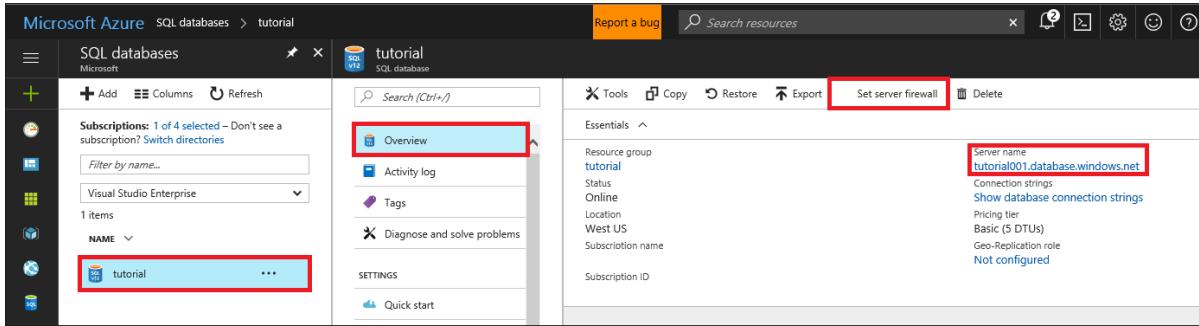
Follow these steps to create a [SQL Database server-level firewall rule](#) for your server to allow connections from a

specific IP address.

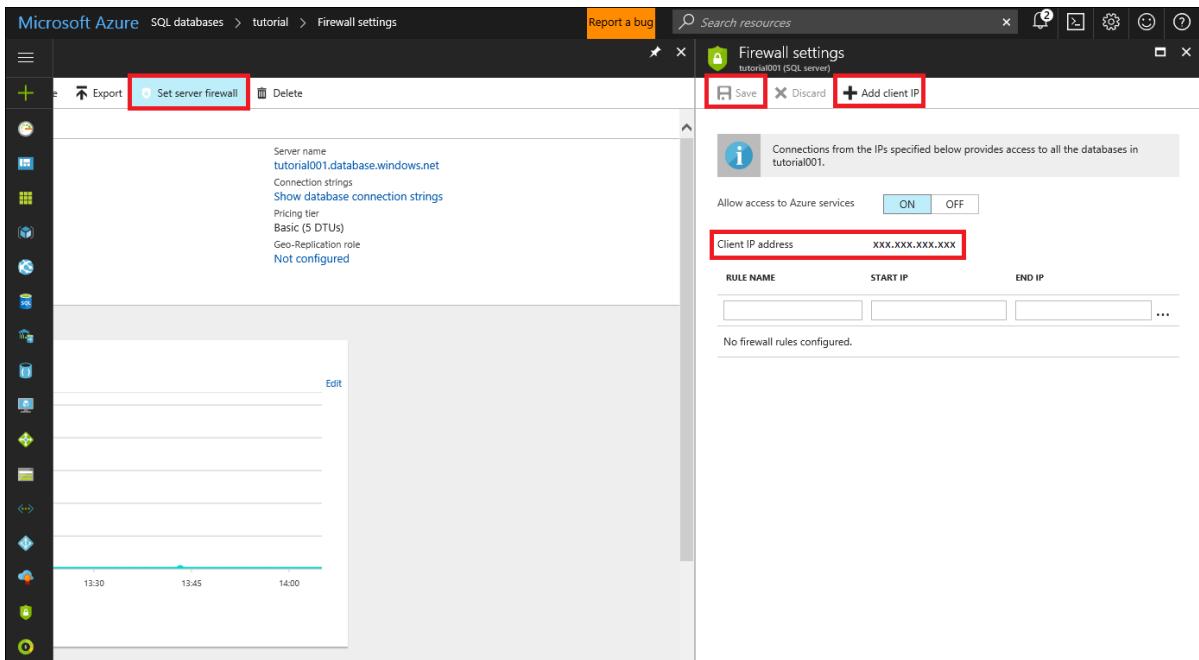
#### NOTE

If you have created a sample database in Azure using one of the previous tutorials or quickstarts and are performing this tutorial on a computer with the same IP address that it had when you walked through those tutorials, you can skip this step as you will have already created a server-level firewall rule.

1. Click **SQL databases** from the left-hand menu and click the database you would like to configure the firewall rule for on the **SQL databases** page. The overview page for your database opens, showing you the fully qualified server name (such as **mynewserver-20170313.database.windows.net**) and provides options for further configuration.



2. Click **Set server firewall** on the toolbar as shown in the previous image. The **Firewall settings** page for the SQL Database server opens.
3. Click **Add client IP** on the toolbar to add the public IP address of the computer connected to the portal with or enter the firewall rule manually and then click **Save**.



4. Click **OK** and then click the **X** to close the **Firewall settings** page.

You can now connect to any database in the server with the specified IP address or IP address range.

## NOTE

SQL Database communicates over port 1433. If you are 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 will not be able to connect to your Azure SQL Database server unless your IT department opens port 1433.

## Create a database-level firewall rule using SSMS

Database-level firewall rules enable you to create different firewall settings for different databases within the same logical server and to create firewall rules that are portable - meaning that they follow the database during a [failover](#) rather than being stored on the SQL server. Database-level firewall rules can only be configured by using Transact-SQL statements and only after you have configured the first server-level firewall rule. For more information, see [Azure SQL Database server-level and database-level firewall rules](#).

Follows these steps to create a database-specific firewall rule.

1. Connect to your database, for example using [SQL Server Management Studio](#).
2. In Object Explorer, right-click on the database you want to add a firewall rule for and click **New Query**. A blank query window opens that is connected to your database.
3. In the query window, modify the IP address to your public IP address and then execute the following query:

```
EXECUTE sp_set_database_firewall_rule N'Example DB Rule','0.0.0.4','0.0.0.4';
```

4. On the toolbar, click **Execute** to create the firewall rule.

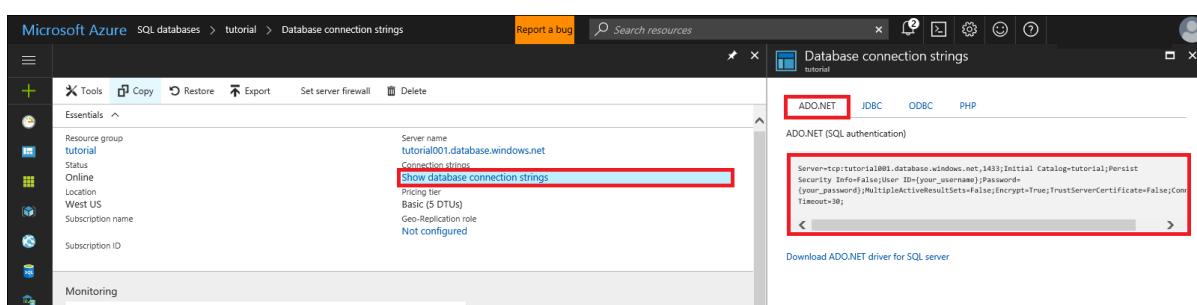
## View how to connect an application to your database using a secure connection string

To ensure a secure, encrypted connection between a client application and SQL Database, the connection string has to be configured to:

- Request an encrypted connection, and
- To not trust the server certificate.

This establishes a connection using Transport Layer Security (TLS) and reduces the risk of man-in-the-middle attacks. You can obtain correctly configured connection strings for your SQL Database for supported client drivers from the Azure portal as shown for ADO.net in this screenshot. For information about TLS and connectivity, see [TLS considerations](#).

1. Select **SQL databases** from the left-hand menu, and click your database on the **SQL databases** page.
2. On the **Overview** page for your database, click **Show database connection strings**.
3. Review the complete **ADO.NET** connection string.



## Creating database users

Before creating any users, you must first choose from one of two authentication types supported by Azure SQL Database:

**SQL Authentication**, which uses username and password for logins and users that are valid only in the context of a specific database within a logical server.

**Azure Active Directory Authentication**, which uses identities managed by Azure Active Directory.

If you want to use [Azure Active Directory](#) to authenticate against SQL Database, a populated Azure Active Directory must exist before you can proceed.

Follow these steps to create a user using SQL Authentication:

1. Connect to your database, for example using [SQL Server Management Studio](#) using your server admin credentials.
2. In Object Explorer, right-click on the database you want to add a new user on and click **New Query**. A blank query window opens that is connected to the selected database.
3. In the query window, enter the following query:

```
CREATE USER ApplicationUser WITH PASSWORD = 'YourStrongPassword1';
```

4. On the toolbar, click **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 to the newly created user, execute the following two commands in a new query window

```
ALTER ROLE db_datareader ADD MEMBER ApplicationUser;
ALTER ROLE db_datawriter ADD MEMBER ApplicationUser;
```

It is best practice to create these non-administrator accounts at the database level to connect to your database unless you need to execute administrator tasks like creating new users. Please review the [Azure Active Directory tutorial](#) on how to authenticate using Azure Active Directory.

## Protect your data with encryption

Azure SQL Database transparent data encryption (TDE) automatically encrypts your data at rest, without requiring any changes to the application accessing the encrypted database. For newly created databases, TDE is on by default. To enable TDE for your database or to verify that TDE is on, follow these steps:

1. Select **SQL databases** from the left-hand menu, and click your database on the **SQL databases** page.
2. Click on **Transparent data encryption** to open the configuration page for TDE.

The screenshot shows the Microsoft Azure portal interface. On the left, the 'SQL databases' blade is open, displaying a list of databases including 'tutorial'. In the center, the 'tutorial - Transparent data encryption' blade is displayed. At the top right of this blade, there is a 'Save' button highlighted with a red box. Below it, the 'Data encryption' section has a switch labeled 'ON' which is also highlighted with a red box. The 'Encryption status' section shows 'Unencrypted'.

3. If necessary, set **Data encryption** to ON and click **Save**.

The encryption process starts in the background. You can monitor the progress by connecting to SQL Database using [SQL Server Management Studio](#) and querying the `encryption_state` column of the `sys.dm_database_encryption_keys` view. A state of 3 indicates that the database is encrypted.

## Enable SQL Database auditing, if necessary

Azure SQL Database 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 could indicate potential security violations. Follow these steps to create a default auditing policy for your SQL database:

1. Select **SQL databases** from the left-hand menu, and click your database on the **SQL databases** page.
2. In the Settings blade, select **Auditing & Threat Detection**. Notice that server-level auditing is disabled and that there is a **View server settings** link that allows you to view or modify the server auditing settings from this context.

The screenshot shows the Microsoft Azure portal interface. On the left, the 'SQL databases' blade is open, displaying a list of databases including 'tutorial'. In the center, the 'tutorial - Auditing & Threat Detection' blade is displayed. The 'Auditing & Threat Detection' section contains the message 'Threat Detection costs \$15/server/month. It will be free for the first 60 days. If Blob Auditing or Threat Detection are enabled on the server, they will always apply to the database, regardless of the database settings.' Below this, the 'Auditing' section shows a switch labeled 'ON' which is highlighted with a red box. The 'Auditing type' section shows 'Blob' selected. The 'Threat Detection' section shows a switch labeled 'ON'.

3. If you prefer to enable an Audit type (or location?) different from the one specified at the server level, turn **ON** Auditing, and choose the **Blob** Auditing Type. If server Blob auditing is enabled, the database-

configured audit will exist side-by-side with the server Blob audit.

4. Select **Storage Details** to open the Audit Logs Storage Blade. Select the Azure storage account where logs will be saved, and the retention period, after which the old logs will be deleted, then click **OK** at the bottom.

**TIP**

Use the same storage account for all audited databases to get the most out of the auditing reports templates.

5. Click **Save**.

**IMPORTANT**

If you want to customize the audited events, you can do this via PowerShell or REST API - see [SQL database auditing](#) for more information.

## Enable SQL Database threat detection

Threat Detection 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 can explore the suspicious events using SQL Database Auditing to determine if they result from an attempt to access, breach or exploit data in the database. Threat Detection 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 example, Threat Detection detects certain anomalous database activities indicating potential SQL injection attempts. SQL injection is one of the common Web application security issues on the Internet, used to attack data-driven applications. Attackers take advantage of application vulnerabilities to inject malicious SQL statements into application entry fields, for breaching or modifying data in the database.

1. Navigate to the configuration blade of the SQL database you want to monitor. In the Settings blade, select **Auditing & Threat Detection**.

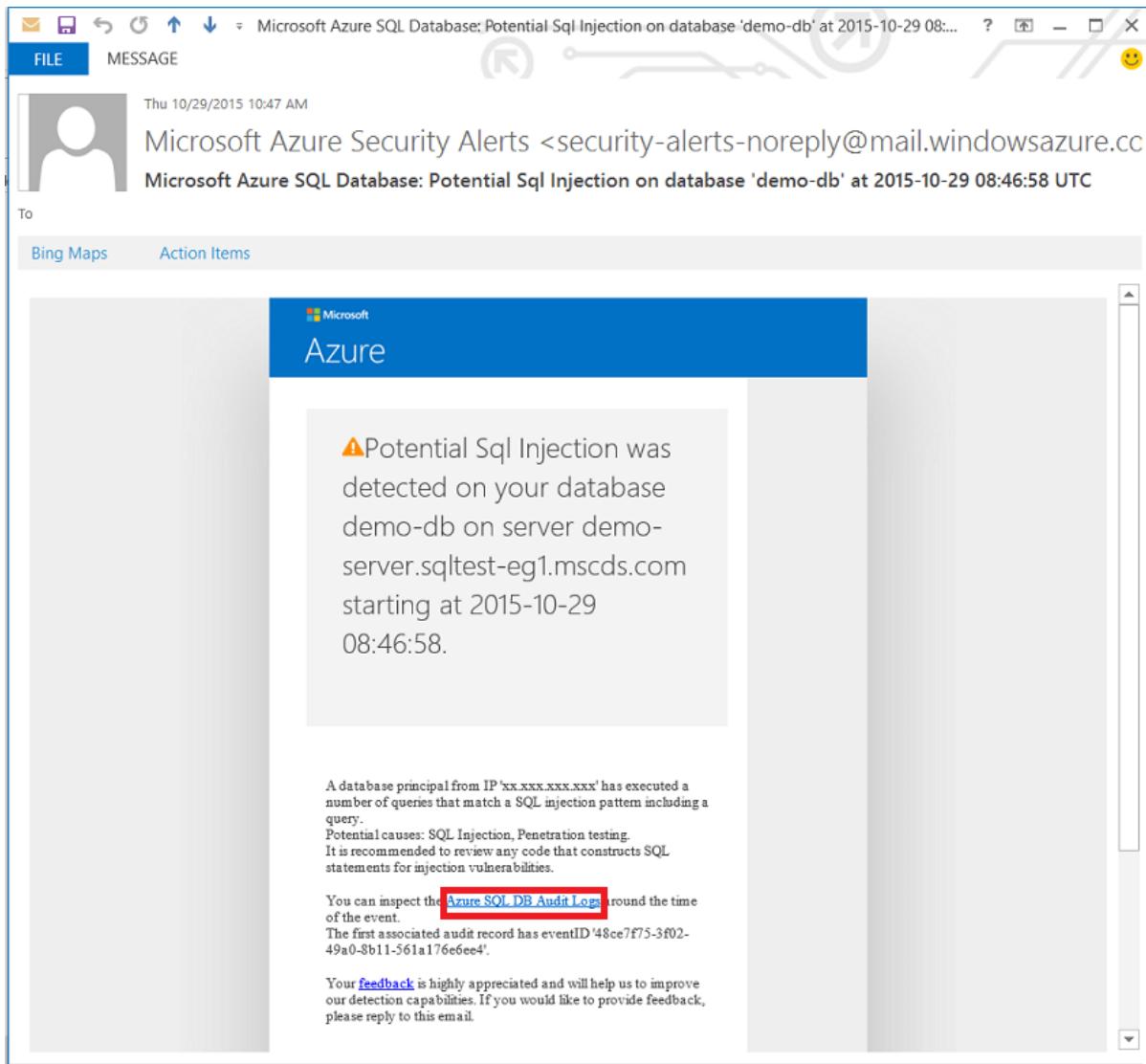
The screenshot shows the Azure portal interface for managing a SQL database's auditing and threat detection settings. On the left, a sidebar lists various database management options like Overview, Activity log, Tags, Diagnose and solve problems, and Settings. Under Settings, the 'Auditing & Threat Detection' option is selected and highlighted in blue. The main content area displays the configuration blade for auditing and threat detection. It includes a summary message about threat detection costs and a note that blob auditing or threat detection must be enabled on the server. Below this, it shows the current state of server-level auditing (Disabled) and threat detection (Disabled). There are toggle switches for 'Auditing' (set to OFF) and 'Threat Detection' (set to OFF). A note indicates that threat detection requires auditing to be turned on. At the top right, there are 'Save', 'View audit logs', and 'Feedback' buttons.

2. In the **Auditing & Threat Detection** configuration blade turn **ON** auditing, which will display the threat detection settings.
3. Turn **ON** threat detection.
4. Configure the list of emails that will receive security alerts upon detection of anomalous database activities.
5. Click **Save** in the **Auditing & Threat detection** blade to save the new or updated auditing and threat detection policy.

This screenshot shows the same configuration blade after the auditing and threat detection policies have been updated. The 'Auditing' switch is now set to 'ON'. The 'Threat Detection (preview)' switch is also set to 'ON'. The 'Send alerts to' section now includes the email address 'Email addresses' and the checked checkbox 'Email service and co-administrators'. A note at the top states that if blob auditing is enabled on the server, it will always apply to the database. The rest of the interface remains largely the same, with sections for auditing type (blob), storage details, and threat detection types.

If anomalous database activities are detected, you will receive an email notification upon detection of anomalous database activities. The email will provide information on the suspicious security event including the nature of the anomalous activities, database name, server name and the event time. In addition, it will

provide information on possible causes and recommended actions to investigate and mitigate the potential threat to the database. The next steps walk you through what to do should you receive such an email:



6. In the email, click on the **Azure SQL Auditing Log** link, which will launch the Azure portal and show the relevant auditing records around the time of the suspicious event.



Showing Audit records up to Thu, 29 Oct 2015 08:46:58 GMT.

EVENT TIME

EVENT ID

PRINCIPAL NAME

EVENT TYPE

ACTION STATUS

Thu, 29 Oct 2015 08:46:58...

48ce7f75-3f02-49a0-8b11...

testlogin

DataAccess

Failure

Thu, 29 Oct 2015 08:46:58...

04e8fb7-b6e0-4fb2-a23a...

testlogin

Login

Success

Thu, 29 Oct 2015 07:36:39...

8bae2f31-aec0-44aa-af63-...

testlogin

DataAccess

Success

Thu, 29 Oct 2015 07:36:36...

4b0f6bd9-71c9-4d25-9a44...

testlogin

DataAccess

Success

Thu, 29 Oct 2015 07:36:34...

9a8b2c5b-ee02-4e28-8db...

testlogin

DataAccess

Success

Thu, 29 Oct 2015 07:36:34...

78bf75f4-4f02-410f-b6ff-c...

testlogin

DataAccess

Success

Thu, 29 Oct 2015 07:36:34...

0bb78eb5-6699-463e-9d8...

testlogin

Login

Success

Thu, 29 Oct 2015 07:36:27...

49ca2098-ab10-4999-995f...

testlogin

DataAccess

Success

Thu, 29 Oct 2015 07:36:20...

2ad9b130-0730-4754-83d...

testlogin

DataAccess

Success

Thu, 29 Oct 2015 07:36:19...

747847f9-0d46-4bb7-8b2...

testlogin

Login

Success

Load more

7. Click on the audit records to view more information on the suspicious database activities such as SQL statement, failure reason and client IP.

The screenshot shows the Audit Record blade with two panes. The left pane displays the audit record details, and the right pane shows the query that triggered the event.

| EVENT TIME       | Thu, 29 Oct 2015 08:46:58 GMT                              |
|------------------|--|
| EVENT ID         | 48ce7f75-3f02-49a0-8b11-561a176e6ee4                       |
| EVENT TYPE       | DataAccess   |
| SERVER NAME      | demo-server.sqltest-eq1.mscds.com                          |
| DATABASE NAME    | demo-db  |
| PRINCIPAL NAME   | testlogin  |
| CLIENT IP        | .Net SqlClient Data Provider                               |
| APPLICATION NAME | Failure  |
| ACTION STATUS    | Err 208, Level 16, State 1, Server demo-server, Line 1 ... |
| FAILURE REASON   | 0  |
| RESPONSE VOLUME  | 0  |
| AFFECTED ROWS    | 0  |
| SERVER DURATION  | 00:00:00.0156359   |

Query pane:

```
1 select * from employee where '1'='1' --' and '1'='1'
```

8. In the Auditing Records blade, click **Open in Excel** to open a pre-configured excel template to import and run deeper analysis of the audit log around the time of the suspicious event.

#### NOTE

In Excel 2010 or later, Power Query and the **Fast Combine** setting is required.

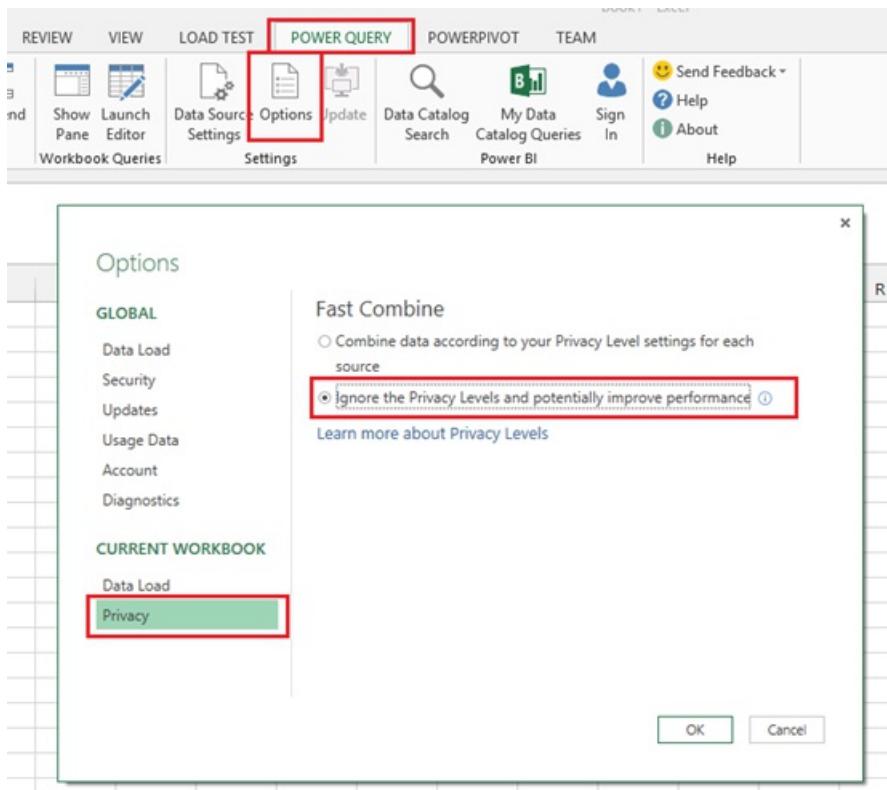


Showing Audit records up to Thu, 29 Oct 2015 08:46:58 GMT.

| EVENT TIME                   | EVENT ID                     | PRINCIPAL NAME | EVENT TYPE | ACTION STATUS |
|------------------------------|------------------------------|----------------|------------|---------------|
| Thu, 29 Oct 2015 08:46:58... | 48ce7f75-3f02-49a0-8b11...   | testlogin      | DataAccess | Failure       |
| Thu, 29 Oct 2015 08:46:58... | 04e8fb7-b6e0-4fb2-a23a...    | testlogin      | Login      | Success       |
| Thu, 29 Oct 2015 07:36:39... | 8baef31-aec0-44aa-af63...    | testlogin      | DataAccess | Success       |
| Thu, 29 Oct 2015 07:36:36... | 4b0f6bd9-71c9-4d25-9a44...   | testlogin      | DataAccess | Success       |
| Thu, 29 Oct 2015 07:36:34... | 9a8b2c5b-ee02-4e28-8db...    | testlogin      | DataAccess | Success       |
| Thu, 29 Oct 2015 07:36:34... | 78bf75f4-4f02-410f-b6ff-c... | testlogin      | DataAccess | Success       |
| Thu, 29 Oct 2015 07:36:34... | 0bb78eb5-6699-463e-9d8...    | testlogin      | Login      | Success       |
| Thu, 29 Oct 2015 07:36:27... | 49ca2098-ab10-4999-995f...   | testlogin      | DataAccess | Success       |
| Thu, 29 Oct 2015 07:36:20... | 2ad9b130-0730-4754-83d...    | testlogin      | DataAccess | Success       |
| Thu, 29 Oct 2015 07:36:19... | 747847f9-0d46-4bb7-8b2...    | testlogin      | Login      | Success       |

[Load more](#)

9. To configure the **Fast Combine** setting - In the **POWER QUERY** ribbon tab, select **Options** to display the Options dialog. Select the Privacy section and choose the second option - 'Ignore the Privacy Levels and potentially improve performance':



10. To load SQL audit logs, ensure that the parameters in the settings tab are set correctly and then select the 'Data' ribbon and click the 'Refresh All' button.



**SQL Audit Log Parameters:** To load SQL audit logs, ensure that the parameters below are set correctly, then select the 'Data' ribbon and click the 'Refresh All' button. The results appear in the "SQL Audit Logs" sheet.

| Parameters                    | Description   | Value                          |
|-------------------------------|---|--------------------------------|
| Auditing Storage Account Name | The name of the Azure storage account that contains the audit logs.   |                                |
| Auditing Storage Table Name   | The name of the Azure table that contains the audit logs (audit log tables start with 'SQLDAuditLogs' prefix). Clear the value to retrieve audit logs from all tables with 'SQLDAuditLogs' prefix (in case the number of retrieved records is equal to the Records Limit value, please specify a specific table name or narrow the Start/End Time range). | SQLDAuditLogsDemoserverDemoddb |
| Start Time                    | The start time for the audit logs to be retrieved (UTC).  | 2015-10-29T08:41:58Z           |
| End Time                      | The end time for the audit logs to be retrieved (UTC).  | 2015-10-29T08:51:58Z           |
| Server Name                   | Allows to retrieve audit logs for specific server. Clear this value to retrieve audit logs for all servers and databases.   |                                |
| Database Name                 | Allows to retrieve audit logs for specific database on a specific server, specified above. Clear this value to retrieve audit logs for all databases on the server specified above.   |                                |
| Records Limit                 | Limits the number of records retrieved from the storage. The default value is 1,000,000 records. The number may be reduced to decrease the refresh time or increased depending on your computer resources (With Excel 2013 64 bits, the workbook was tested with up to 6,000,000 records).  | 1,000,000                      |

**Excel Prerequisites:** Excel 2010 or later, Power Query and configured 'Fast Combine' setting (see below for details).

11. The results appear in the **SQL Audit Logs** sheet which enables you to run deeper analysis of the anomalous activities that were detected, and mitigate the impact of the security event in your application.

## Next steps

In this tutorial, you learned to improve the protection of your database against malicious users or unauthorized access with just a few simple steps. You learned how to:

- Set up firewall rules for your server and or database
- Connect to your database using a secure connection string
- Manage user access
- Protect your data with encryption
- Enable SQL Database auditing
- Enable SQL Database threat detection

Advance to the next tutorial to learn how to implement a geo-distributed database.

[Implement a geo-distributed database](#)

# Implement a geo-distributed database

9/25/2018 • 9 minutes to read • [Edit Online](#)

In this tutorial, you configure an Azure SQL database and application for failover to a remote region, and then test your failover plan. You learn how to:

- Create database users and grant them permissions
- Set up a database-level firewall rule
- Create a [geo-replication failover group](#)
- Create and compile a Java application to query an Azure SQL database
- Perform a disaster recovery drill

If you don't have an Azure subscription, [create a free account](#) before you begin.

## Prerequisites

To complete this tutorial, make sure the following prerequisites are completed:

- Installed the latest [Azure PowerShell](#).
- Installed an Azure SQL database. This tutorial uses the AdventureWorksLT sample database with a name of **mySampleDatabase** from one of these quick starts:
  - [Create DB - Portal](#)
  - [Create DB - CLI](#)
  - [Create DB - PowerShell](#)
- Have identified a method to execute SQL scripts against your database, you can use one of the following query tools:
  - The query editor in the [Azure portal](#). For more information on using the query editor in the Azure portal, see [Connect and query using Query Editor](#).
  - The newest version of [SQL Server Management Studio](#), which is an integrated environment for managing any SQL infrastructure, from SQL Server to SQL Database for Microsoft Windows.
  - The newest version of [Visual Studio Code](#), which is a graphical code editor for Linux, macOS, and Windows that supports extensions, including the [mssql extension](#) for querying Microsoft SQL Server, Azure SQL Database, and SQL Data Warehouse. For more information on using this tool with Azure SQL Database, see [Connect and query with VS Code](#).

## Create database users and grant permissions

Connect to your database and create user accounts using one of the following query tools:

- The Query editor in the Azure portal
- SQL Server Management Studio
- Visual Studio Code

These user accounts replicate automatically to your secondary server (and be kept in sync). To use SQL Server Management Studio or Visual Studio Code, you may need to configure a firewall rule if you are connecting from a client at an IP address for which you have not yet configured a firewall. For detailed steps, see [Create a server-level firewall rule](#).

- In a query window, execute the following query to create two user accounts in your database. This script

grants **db\_owner** permissions to the **app\_admin** account and grants **SELECT** and **UPDATE** permissions to the **app\_user** account.

```
CREATE USER app_admin WITH PASSWORD = 'ChangeYourPassword1';
--Add SQL user to db_owner role
ALTER ROLE db_owner ADD MEMBER app_admin;
--Create additional SQL user
CREATE USER app_user WITH PASSWORD = 'ChangeYourPassword1';
--grant permission to SalesLT schema
GRANT SELECT, INSERT, DELETE, UPDATE ON SalesLT.Product TO app_user;
```

## Create database-level firewall

Create a [database-level firewall rule](#) for your SQL database. This database-level firewall rule replicates automatically to the secondary server that you create in this tutorial. For simplicity (in this tutorial), use the public IP address of the computer on which you are performing the steps in this tutorial. To determine the IP address used for the server-level firewall rule for your current computer, see [Create a server-level firewall](#).

- In your open query window, replace the previous query with the following query, replacing the IP addresses with the appropriate IP addresses for your environment.

```
-- Create database-level firewall setting for your public IP address
EXECUTE sp_set_database_firewall_rule @name = N'myGeoReplicationFirewallRule',@start_ip_address =
'0.0.0.0', @end_ip_address = '0.0.0.0';
```

## Create an active geo-replication auto failover group

Using Azure PowerShell, create an [active geo-replication auto failover group](#) between your existing Azure SQL server and the new empty Azure SQL server in an Azure region, and then add your sample database to the failover group.

### IMPORTANT

These cmdlets require Azure PowerShell 4.0. This sample requires the Azure PowerShell module version 5.1.1 or later. Run `Get-Module -ListAvailable AzureRM` to find the version. If you need to install or upgrade, see [Install Azure PowerShell module](#). Run `Connect-AzureRmAccount` to create a connection with Azure.

1. Populate variables for your PowerShell scripts using the values for your existing server and sample database, and provide a globally unique value for failover group name.

```
$adminlogin = "ServerAdmin"
$password = "ChangeYourAdminPassword1"
$myresourcegroupname = "<your resource group name>"
$mylocation = "<your resource group location>"
$myservername = "<your existing server name>"
$mydatabasename = "mySampleDatabase"
$mydrlocation = "<your disaster recovery location>"
$mydrservername = "<your disaster recovery server name>"
$myfailovergroupname = "<your unique failover group name>"
```

2. Create an empty backup server in your failover region.

```
$mydrserver = New-AzureRmSqlServer -ResourceGroupName $myresourcegroupname ` 
-ServerName $mydrservername ` 
-Location $mydrlocation ` 
-SqlAdministratorCredentials $(New-Object -TypeName System.Management.Automation.PSCredential - 
ArgumentList $adminlogin, $(ConvertTo-SecureString -String $password -AsPlainText -Force)) 
$mydrserver
```

3. Create a failover group between the two servers.

```
$myfailovergroup = New-AzureRmSqlDatabaseFailoverGroup ` 
-ResourceGroupName $myresourcegroupname ` 
-ServerName $myservername ` 
-PartnerServerName $mydrservername ` 
-FailoverGroupName $myfailovergroupname ` 
-FailoverPolicy Automatic ` 
-GracePeriodWithDataLossHours 2 
$myfailovergroup
```

4. Add your database to the failover group.

```
$myfailovergroup = Get-AzureRmSqlDatabase ` 
-ResourceGroupName $myresourcegroupname ` 
-ServerName $myservername ` 
-DatabaseName $mydatabasename | ` 
Add-AzureRmSqlDatabaseToFailoverGroup ` 
-ResourceGroupName $myresourcegroupname ` 
-ServerName $myservername ` 
-FailoverGroupName $myfailovergroupname 
$myfailovergroup
```

## Install Java software

The steps in this section assume that you are familiar with developing using Java and are new to working with Azure SQL Database.

### Mac OS

Open your terminal and navigate to a directory where you plan on creating your Java project. Install **brew** and **Maven** by entering the following commands:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)" 
brew update 
brew install maven
```

For detailed guidance on installing and configuring Java and Maven environment, go the [Build an app using SQL Server](#), select **Java**, select **MacOS**, and then follow the detailed instructions for configuring Java and Maven in step 1.2 and 1.3.

### Linux (Ubuntu)

Open your terminal and navigate to a directory where you plan on creating your Java project. Install **Maven** by entering the following commands:

```
sudo apt-get install maven
```

For detailed guidance on installing and configuring Java and Maven environment, go the [Build an app using SQL Server](#), select **Java**, select **Ubuntu**, and then follow the detailed instructions for configuring Java and Maven in

step 1.2, 1.3, and 1.4.

## Windows

Install [Maven](#) using the official installer. Use Maven to help manage dependencies, build, test, and run your Java project. For detailed guidance on installing and configuring Java and Maven environment, go the [Build an app using SQL Server](#), select **Java**, select Windows, and then follow the detailed instructions for configuring Java and Maven in step 1.2 and 1.3.

## Create SqldbSample project

1. In the command console (such as Bash), create a Maven project.

```
bash mvn archetype:generate "-DgroupId=com.sql dbsamples" "-DartifactId=SqlDbSample" "-DarchetypeArtifactId=maven-archetype-quickstart" "-Dversion=1.0.0"
```

2. Type **Y** and click **Enter**.

3. Change directories into your newly created project.

```
cd SqlDbSamples
```

4. Using your favorite editor, open the pom.xml file in your project folder.

5. Add the Microsoft JDBC Driver for SQL Server dependency to your Maven project by opening your favorite text editor and copying and pasting the following lines into your pom.xml file. Do not overwrite the existing values prepopulated in the file. The JDBC 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 version of Java to compile the project against by adding the following "properties" section into the pom.xml file after the "dependencies" section.

```
<properties>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

7. Add the following "build" section into the pom.xml file after the "properties" section to support manifest files in jars.

```

<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 (C:\apache-maven-3.5.0\SqlDbSample\src\main\java\com\sql dbsamples\App.java) and replace the contents with the following contents. Replace the failover group name with the name for your failover group. If you have changed the values for the database name, user, or password, change those values as well.

```

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 = "myfailovergroupname";

    private static final String DB_NAME = "mySampleDatabase";
    private static final String USER = "app_user";
    private static final String PASSWORD = "ChangeYourPassword1";

    private static final String READ_WRITE_URL =
String.format("jdbc:sqlserver://%s.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://%s.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("#####
##### 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"));
                Thread.sleep(1000);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static boolean insertData(int id) {
        Connection conn = null;
        Statement stmt = null;
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            conn = DriverManager.getConnection(READ_WRITE_URL);
            stmt = conn.createStatement();
            ps = conn.prepareStatement("INSERT INTO MyTable VALUES (" + id + ")");
            ps.executeUpdate();
            return true;
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (rs != null) {
                try {
                    rs.close();
                } catch (Exception e) {
                }
            }
            if (ps != null) {
                try {
                    ps.close();
                } catch (Exception e) {
                }
            }
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (Exception e) {
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (Exception e) {
                }
            }
        }
        return false;
    }
}

```

```

        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 that we can use to find the product
    again later
    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 that was 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 that is 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.

## Compile and run the SqlDbSample project

1. In the command console, execute to following command.

```
mvn package
```

2. When finished, execute the following command to run the application (it runs for about 1 hour unless you stop it manually):

```
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
```

## Perform disaster recovery drill

1. Call manual failover of failover group.

```
Switch-AzureRMSqlDatabaseFailoverGroup `  
-ResourceGroupName $myresourcegroupname `  
-ServerName $mydrservername `  
-FailoverGroupName $myfailovergroupname
```

2. Observe the application results during failover. Some inserts fail while the DNS cache refreshes.
3. Find out which role your disaster recovery server is performing.

```
$mydrserver.ReplicationRole
```

4. Fallback.

```
Switch-AzureRMSqlDatabaseFailoverGroup `  
-ResourceGroupName $myresourcegroupname `  
-ServerName $myservername `  
-FailoverGroupName $myfailovergroupname
```

5. Observe the application results during fallback. Some inserts fail while the DNS cache refreshes.
6. Find out which role your disaster recovery server is performing.

```
$fileovergroup = Get-AzureRMSqlDatabaseFailoverGroup `  
-FailoverGroupName $myfailovergroupname `  
-ResourceGroupName $myresourcegroupname `  
-ServerName $mydrservername  
$fileovergroup.ReplicationRole
```

## Next steps

In this tutorial, you learned to configure an Azure SQL database and application for failover to a remote region, and then test your failover plan. You learned how to:

- Create database users and grant them permissions

- Set up a database-level firewall rule
- Create a geo-replication failover group
- Create and compile a Java application to query an Azure SQL database
- Perform a disaster recovery drill

Advance to the next tutorial to migrate SQL Server to Azure SQL Database Managed Instance using DMS.

[Migrate SQL Server to Azure SQL Database Managed Instance using DMS](#)

# Migrate SQL Server to Azure SQL Database Managed Instance offline using DMS

10/23/2018 • 8 minutes to read • [Edit Online](#)

You can use the 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 an Azure SQL Database Managed Instance by using the Azure Database Migration Service.

In this tutorial, you learn how to:

- Create an instance of the Azure Database Migration Service.
- Create a migration project by using the 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 Azure SQL Database Managed Instance. For an online migration, see [Migrate SQL Server to Azure SQL Database Managed Instance online using DMS](#).

## Prerequisites

To complete this tutorial, you need to:

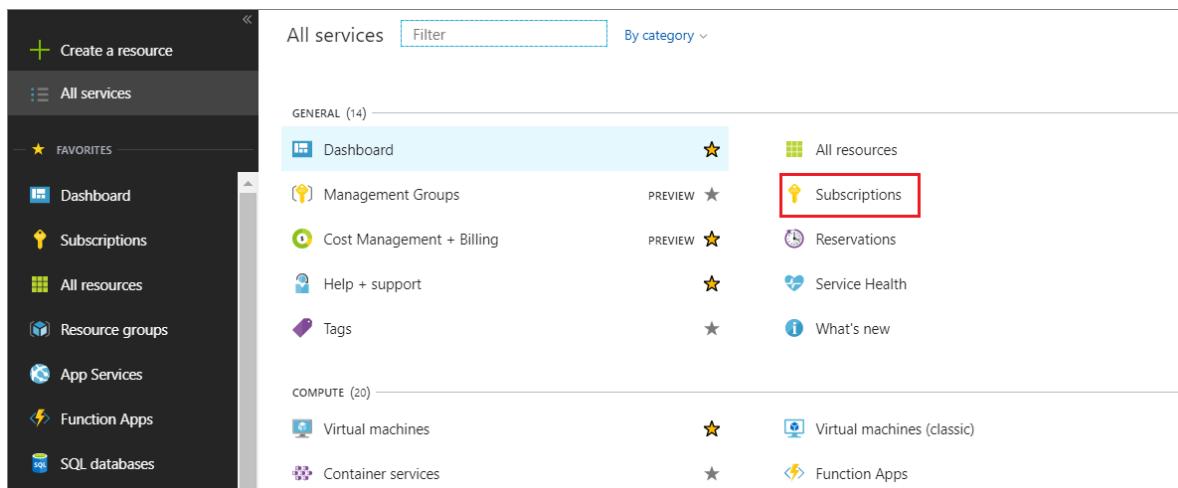
- Create a VNET for the Azure Database Migration Service by using the Azure Resource Manager deployment model, which provides site-to-site connectivity to your on-premises source servers by using either [ExpressRoute](#) or [VPN](#). [Learn network topologies for Azure SQL DB Managed Instance migrations using the Azure Database Migration Service](#).
- Ensure that your Azure Virtual Network (VNET) Network Security Group rules don't block the following communication ports 443, 53, 9354, 445, 12000. For more detail on Azure VNET 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 the 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 the 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 the 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.
- Create a network share that the 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. The 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 the 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**.



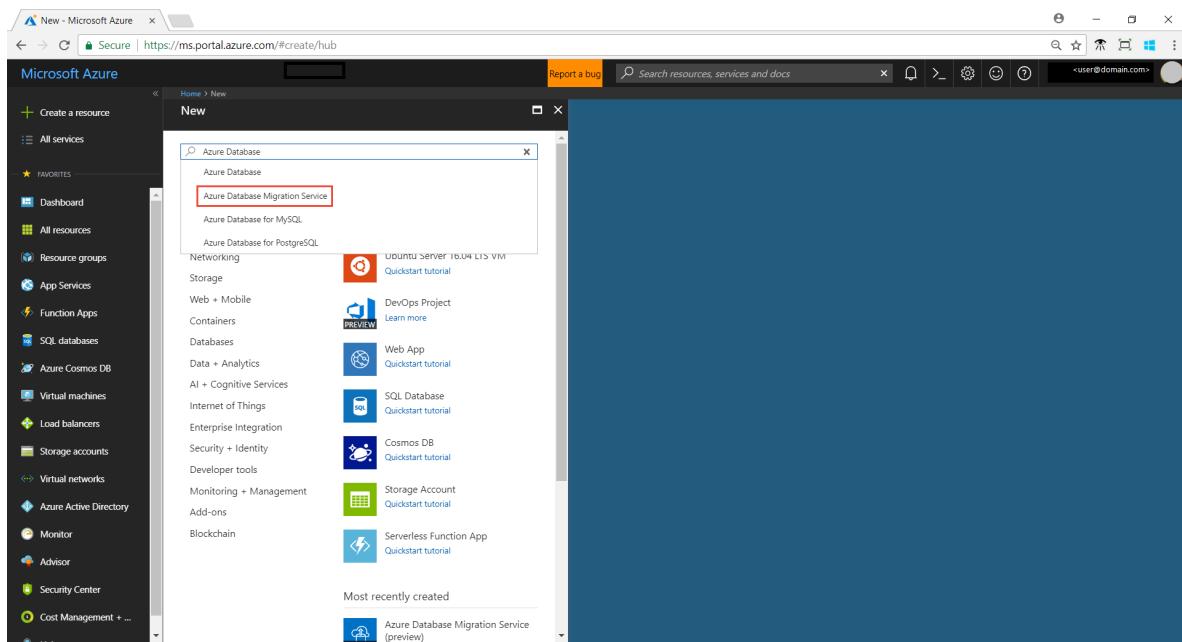
2. Select the subscription in which you want to create the instance of the Azure Database Migration Service, and then select **Resource providers**.

3. Search for migration, and then to the right of **Microsoft.DataMigration**, select **Register**.

| Provider                | Status         | Action          |
|-------------------------|----------------|-----------------|
| Microsoft.DataMigration | Not Registered | <b>Register</b> |

## 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 | <a href="#">Documentation</a><br><a href="#">Privacy Statement</a> |

**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 (VNET) or create one.

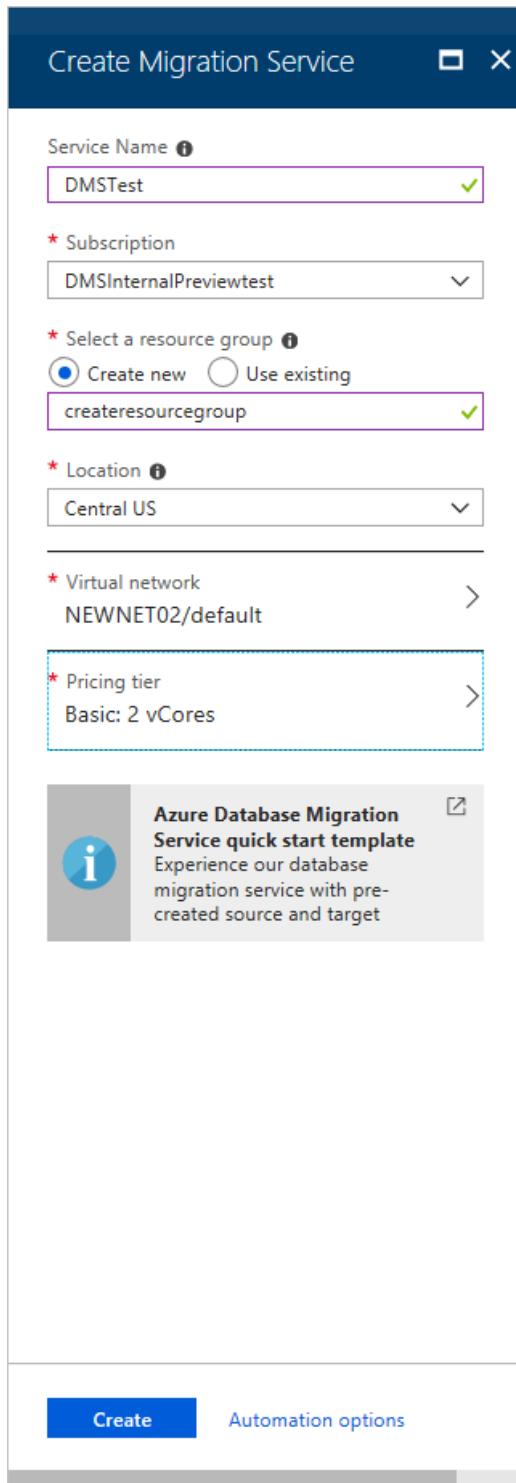
The VNET provides the Azure Database Migration Service with access to the source SQL Server and target Azure SQL Database Managed Instance.

For more information on how to create a VNET 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 the Azure Database Migration Service](#).

6. Select a pricing tier.

For more information on costs and pricing tiers, see the [pricing page](#).

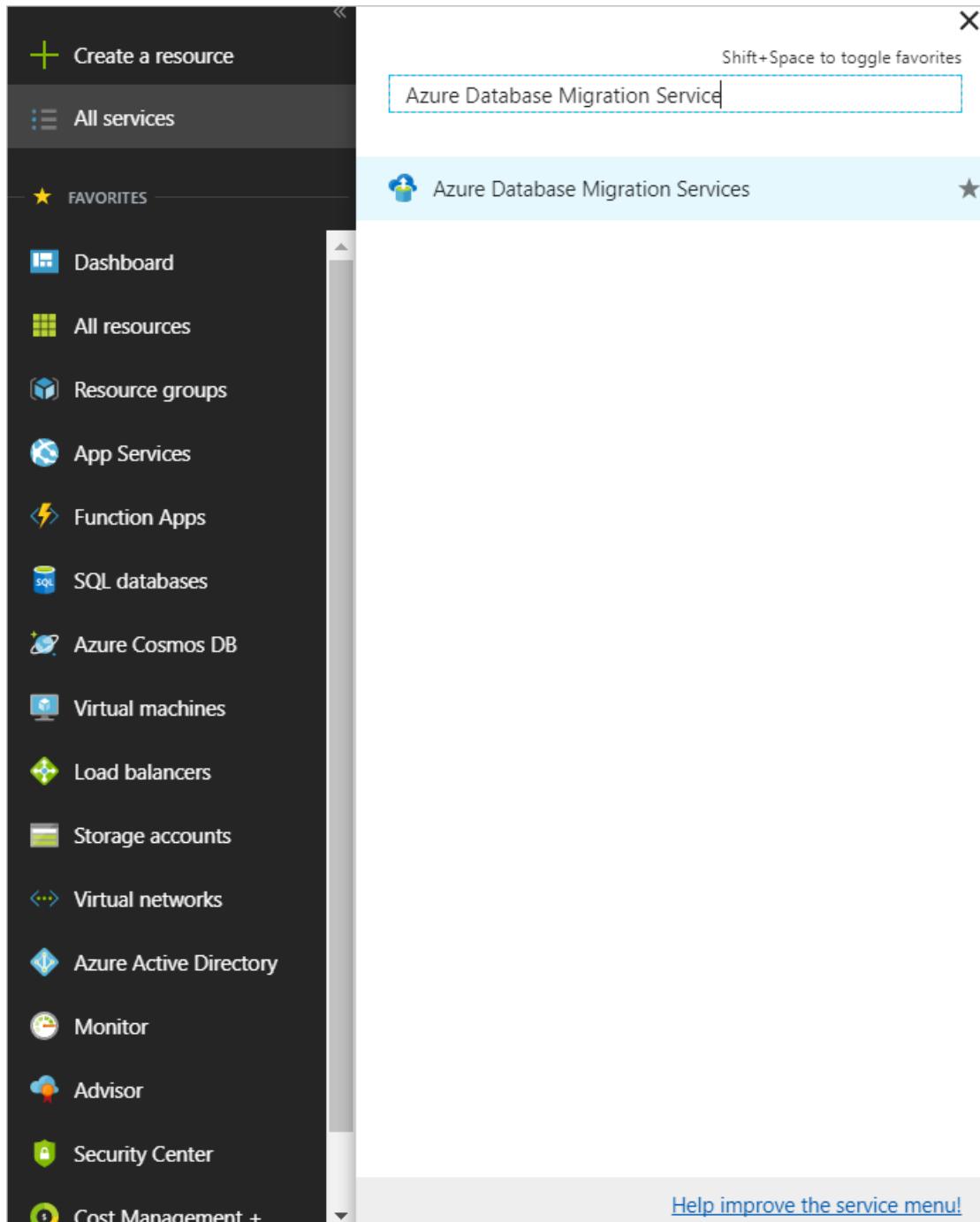


7. Select **Create** to create the service.

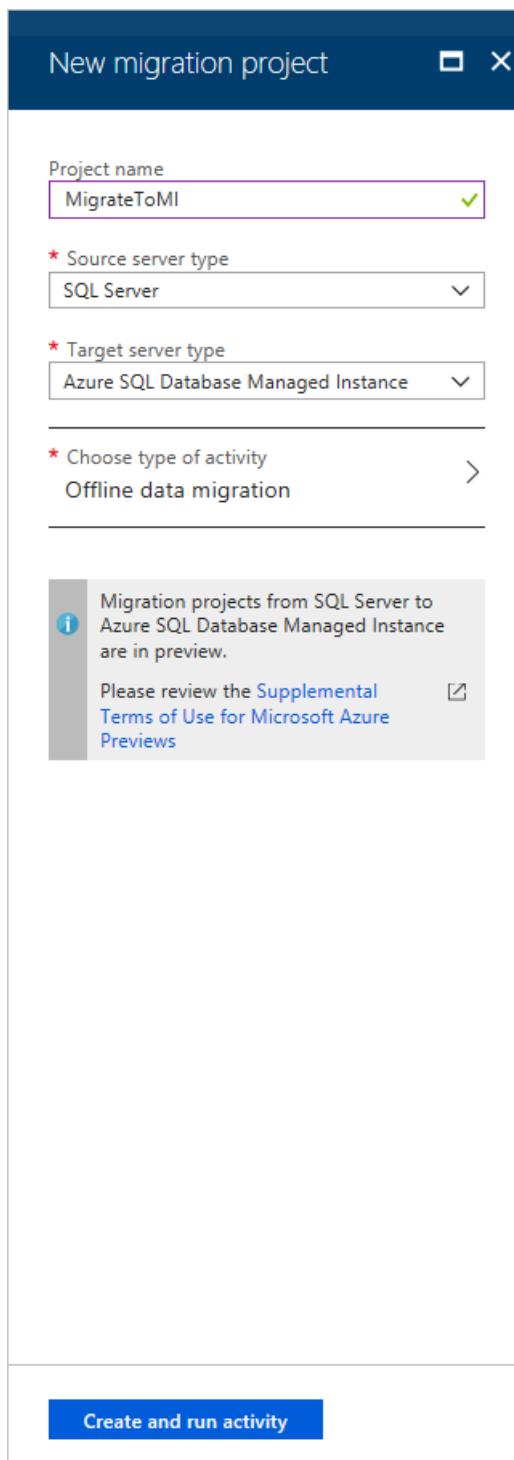
# Create a migration project

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

1 Select source >

2 Select databases >

3 Select target >

4 Summary >

### Source details

\* Source SQL Server instance name [i](#)  
Servername.domainname.com

Authentication type  
SQL Authentication

\* User Name [i](#)  
Enter user name

Password  
Enter password

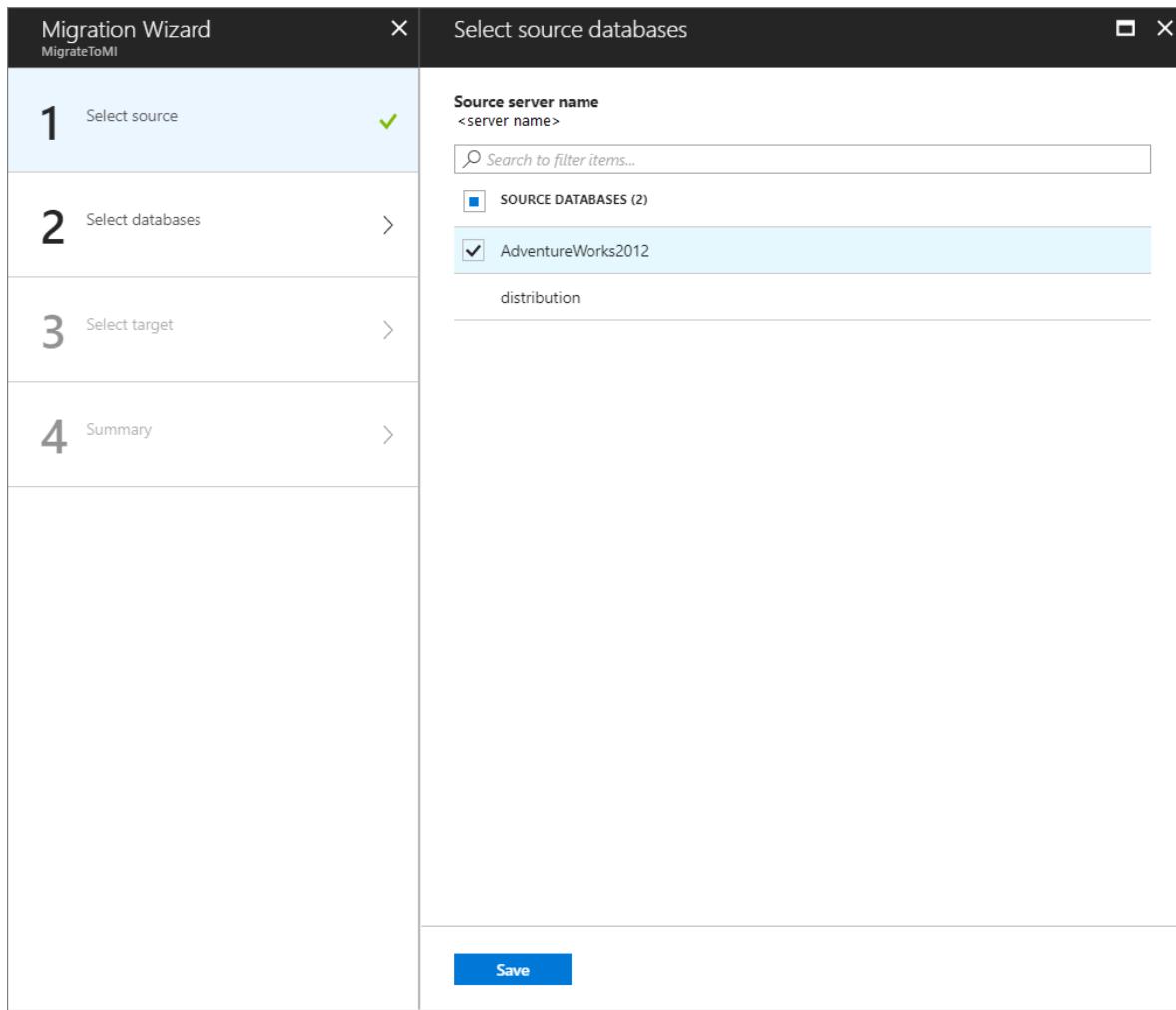
Connection properties

Encrypt connection

Trust server certificate

**Save**

3. Select **Save**.
4. On the **Select source databases** screen, select the **Adventureworks2012** database for migration.

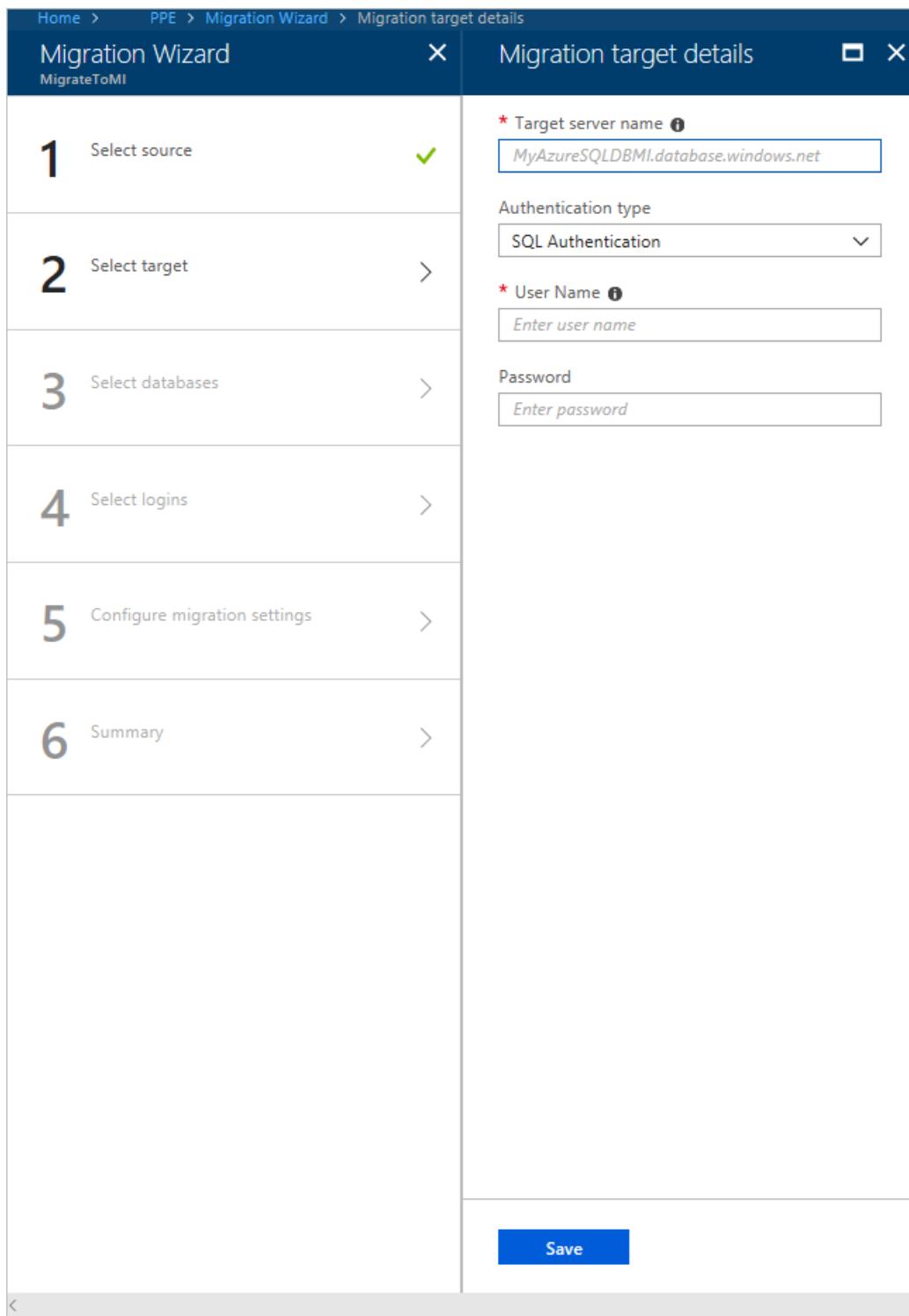


5. Select **Save**.

## 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.

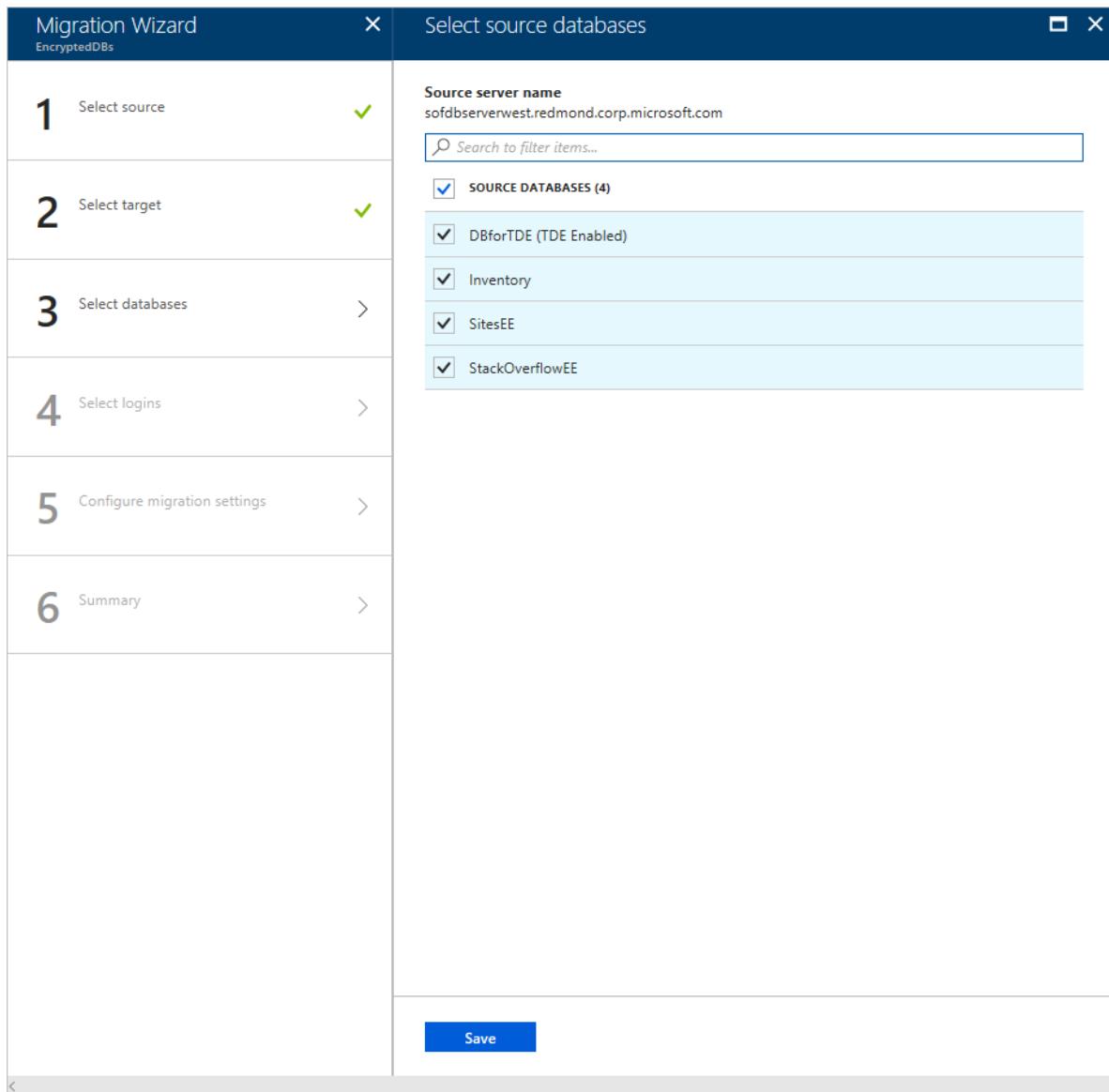
If you haven't already provisioned the Azure SQL Database Managed Instance, select **No** for a link to help you provision the instance. You can still proceed with project creation and then, when the Azure SQL Database Managed Instance is ready, return to this specific project to execute the migration.



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

This release only supports migrating the SQL 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\SQLIaaSExtensi...                          | 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_PolicyTsqlExecution...                           | SQL        | master           | Disabled | Login '#MS_PolicyTsqlExecutionLogin##' 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"/> sqluser           | SQL        | master           | Enabled  |  |
| <input checked="" type="checkbox"/> distributor_admin | 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.                            |
| DEMOSSERVER\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 SERVICE\SQLWriter                                  | Windows    | master           | Enabled  | LoginType 'WindowsUser' is not supported for migration.                            |
| NT_Cardinal\Collation                                 | Windows    | master           | Enabled  | 1 maintenance 'WindowsCollation' 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 the 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. The 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 <a href="#">User Account Control</a> 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 the Azure Database Migration Service with access to your storage account container to which the service uploads the back-up 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 are 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 DB Managed Instance provisioned from the drop-down menu. Select the target <b>Azure SQL Database Managed Instance</b> in the drop-down menu. |

Migration Wizard

EncryptedDBs

1 Select source ✓

2 Select target ✓

3 Select databases ✓

4 Select logins ✓

5 Configure migration settings >

6 Summary >

Configure migration settings

Choose source backup option

I will let Azure Database Migration Service create backup files.

Backup settings

Ensure that the service account running the source SQL Server instance has write privileges on the network location share that you created.

\* Network share location that Azure Database Migration Service can take database backups to  
\\Networkserver.domain.corp.contoso.com ✓

Make sure the Windows user has full control privilege on the network share that you created above. The 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, the Windows user must be the built-in administrator account and User Account Control must be disabled for Azure Database Migration Service to upload, copy and delete the certificates files.)

\* Windows User Azure Database Migration Service impersonates to upload files to Azure Storage  
DomainUsername ✓

Password  
\*\*\*\*\* ✓

Storage account settings

Provide the SAS URI that allows Azure Database Migration Service to access your storage account container that Azure Database Migration Service will upload the backup files to and use for migrating the databases to SQL DB Managed instance. Use this [link](#) for creating SAS URI, make sure to select all permissions (Read, Write, Delete and List)

\* SAS URI for Azure Storage container that Azure Database Migration Service will upload the files to  
sig=i7sTHpcHt7x%2FqP1J0HadK5hE%2F%2Fm%2B3%2B%2Bf%2F%2BpuQYzxl%3D&sr=c ✓

TDE Settings

Please select the target Azure SQL Database Managed Instance. Ensure that you have write permission for target server. Azure Database Migration Service requires this for migration of Transparent Data Encryption (TDE) enabled databases.

\* Select subscription containing the target Azure SQL Database Managed Instance server  
DMSINTERNALDEMO

\* Select target Azure SQL Database Managed Instance  
demomi

Advanced settings ▾

Save

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. Step 1 has a green checkmark. Steps 2 through 5 have green checkmarks. 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.ne       |
| Target server version | Azure SQL Database Managed Instance<br>12.0.2000.8 |
| Source server name    | 10.105.129.164                                     |
| Source server version | SQL Server 2012<br>11.0.7462.6                     |
| Databases to migrate  | 2 of 5   |
| Login(s) to migrate   | 6/28   |
| Validation option     | 1/1 options selected                               |

At the bottom of the 'Migration summary' window is a blue button 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

**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

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

**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

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         |

**Logins**

1

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](#).

# Set up SQL Data Sync to sync data between Azure SQL Database and SQL Server on-premises

10/15/2018 • 12 minutes to read • [Edit Online](#)

In this tutorial, you learn how to set up Azure SQL Data Sync by creating a hybrid sync group that contains both Azure SQL Database and SQL Server instances. The new sync group is fully configured and synchronizes on the schedule you set.

This tutorial assumes that you have at least some prior experience with SQL Database and with SQL Server.

For an overview of SQL Data Sync, see [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#).

For complete PowerShell examples that show how to configure SQL Data Sync, see the following articles:

- [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](#)

## Step 1 - Create sync group

### Locate the Data Sync settings

1. In your browser, navigate to the Azure portal.
2. In the portal, locate your SQL databases from your Dashboard or from the SQL Databases icon on the toolbar.

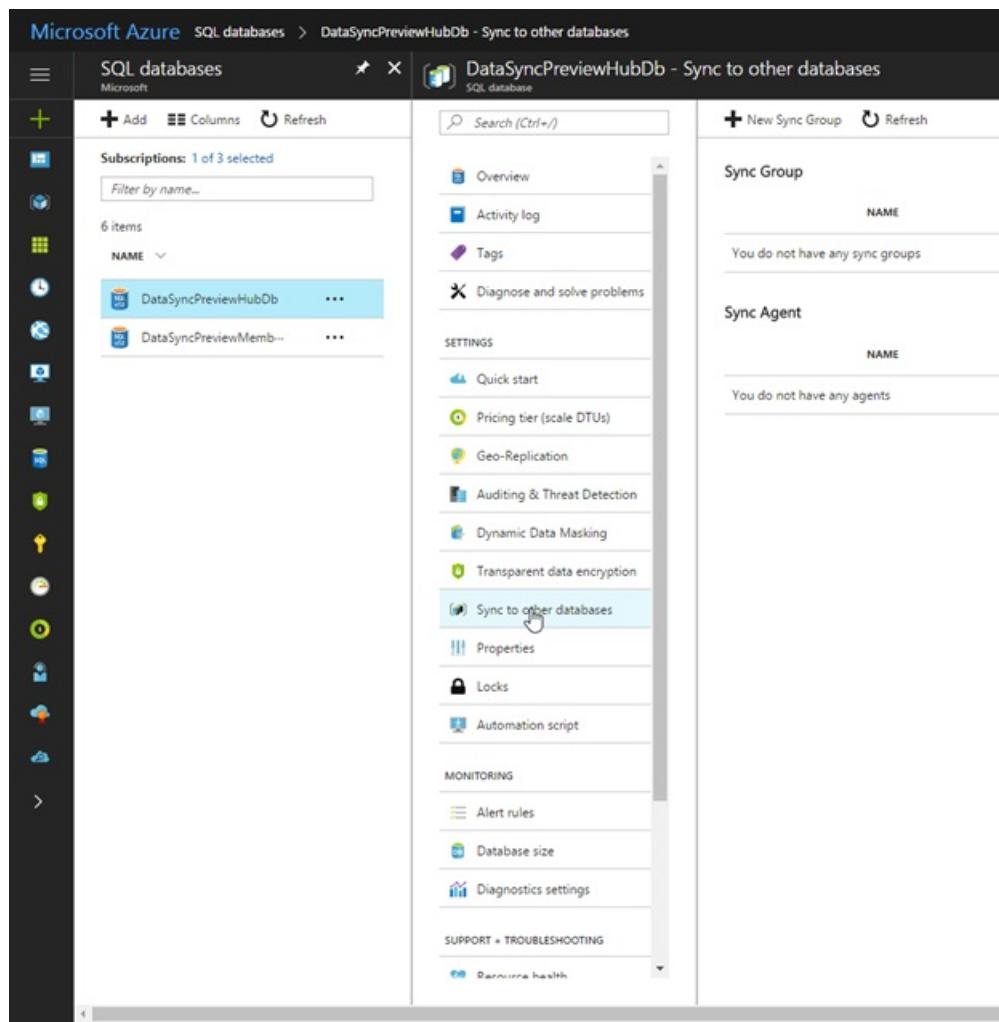
The screenshot shows the Microsoft Azure SQL databases portal. The top navigation bar says "Microsoft Azure SQL databases". Below it, there's a toolbar with icons for "Add", "Columns", and "Refresh". A sidebar on the left has icons for "SQL databases", "Subscriptions", "Metrics", "Logs", "Auditing", "Encryption", "Backup", and "Collation". The main content area shows a table with the following data:

| NAME                    | STATUS |
|-------------------------|--------|
| DataSyncPreviewHubDb    | Online |
| DataSyncPreviewMemberDb | Online |

3. On the **SQL databases** page, select the existing SQL database that you want to use as the hub database for Data Sync. The SQL database page opens.

The hub database is the central endpoint of the sync topology, in which a sync group has multiple database endpoints. All other database endpoints in the same sync group - that is, all member databases - sync with the hub database.

4. On the SQL database page for the selected database, select **Sync to other databases**. The Data Sync page opens.



## Create a new Sync Group

1. On the Data Sync page, select **New Sync Group**. The **New sync group** page opens with Step 1, **Create sync group**, highlighted. The **Create Data Sync Group** page also opens.
2. On the **Create Data Sync Group** page, do the following things:
  - a. In the **Sync Group Name** field, enter a name for the new sync group.
  - b. In the **Sync Metadata Database** section, choose whether to create a new database (recommended) or to use an existing database.

### NOTE

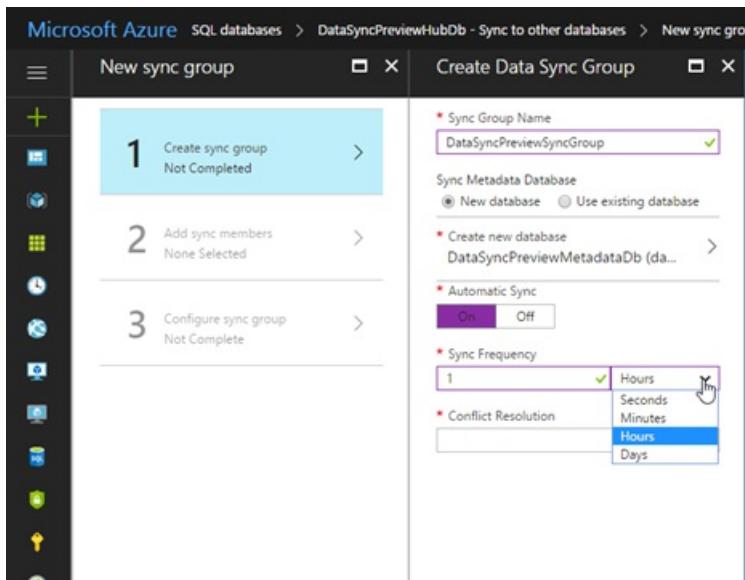
Microsoft recommends that you create a new, empty database to use as the Sync Metadata Database. Data Sync creates tables in this database and runs a frequent workload. This database is automatically shared as the Sync Metadata Database for all of your Sync groups in the selected region. You can't change the Sync Metadata Database or its name without removing all the Sync Groups and Sync Agents in the region.

If you chose **New database**, select **Create new database**. The **SQL Database** page opens. On the **SQL Database** page, name and configure the new database. Then select **OK**.

If you chose **Use existing database**, select the database from the list.

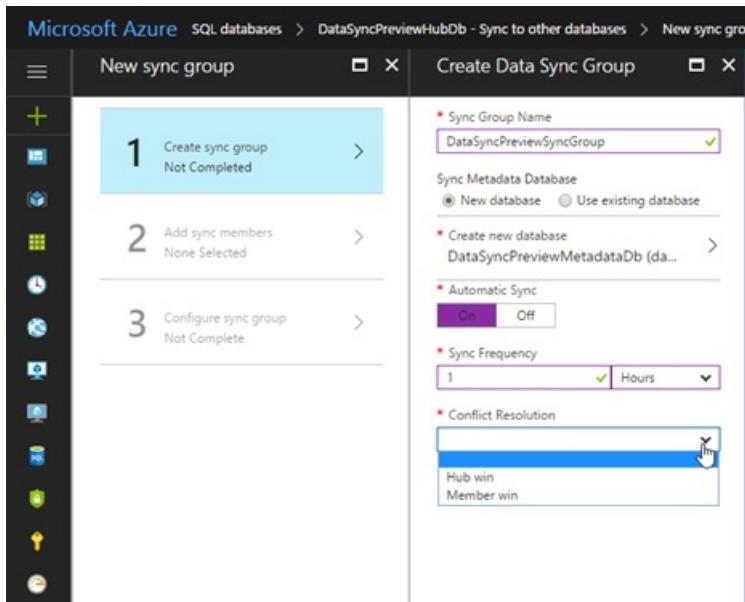
- c. In the **Automatic Sync** section, first select **On** or **Off**.

If you chose **On**, in the **Sync Frequency** section, enter a number and select Seconds, Minutes, Hours, or Days.



- d. In the **Conflict Resolution** section, select "Hub wins" or "Member wins."

"Hub wins" means that, when a conflict occurs, the data in the hub database overwrites the conflicting data in the member database. "Member wins" means that, when a conflict occurs, the data in the member database overwrites the conflicting data in the hub database.

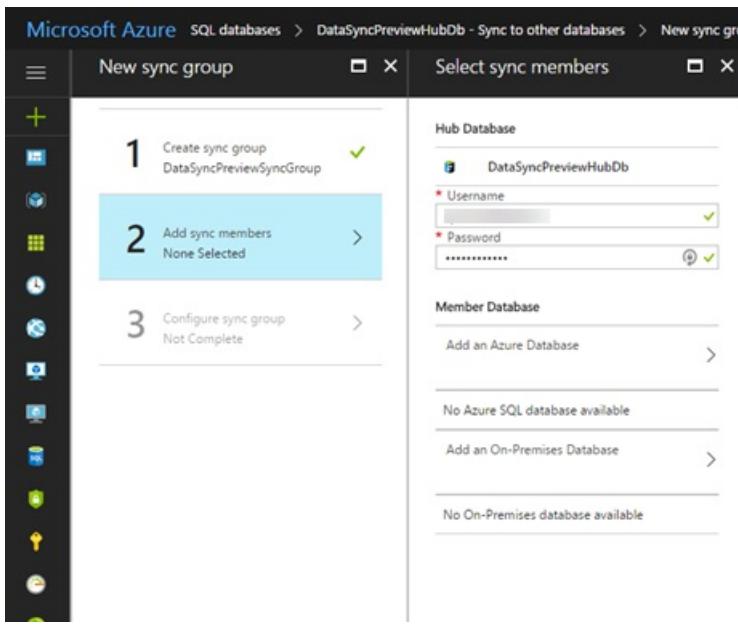


- e. Select **OK** and wait for the new sync group to be created and deployed.

## Step 2 - Add sync members

After the new sync group is created and deployed, Step 2, **Add sync members**, is highlighted in the **New sync group** page.

In the **Hub Database** section, enter the existing credentials for the SQL Database server on which the hub database is located. Don't enter *new* credentials in this section.

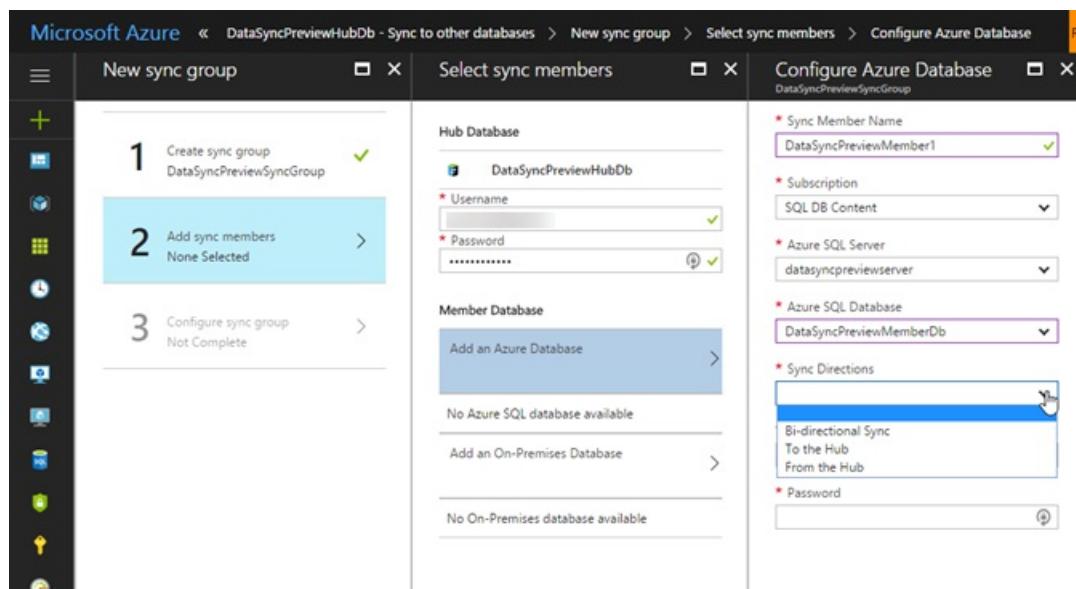


## Add an Azure SQL Database

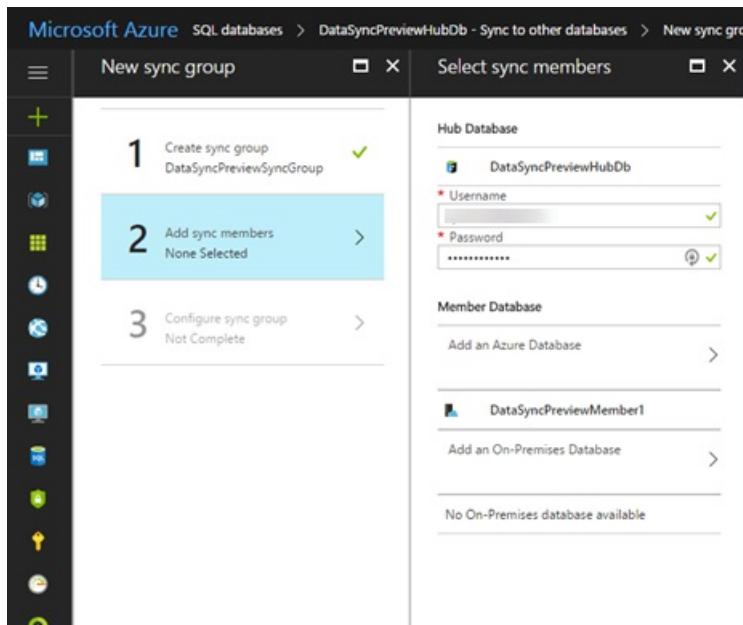
In the **Member Database** section, optionally add an Azure SQL Database to the sync group by selecting **Add an Azure Database**. The **Configure Azure Database** page opens.

On the **Configure Azure Database** page, do the following things:

1. 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.
2. In the **Subscription** field, select the associated Azure subscription for billing purposes.
3. In the **Azure SQL Server** field, select the existing SQL database server.
4. In the **Azure SQL Database** field, select the existing SQL database.
5. In the **Sync Directions** field, select Bi-directional Sync, To the Hub, or From the Hub.



6. In the **Username** and **Password** fields, enter the existing credentials for the SQL Database server on which the member database is located. Don't enter *new* credentials in this section.
7. Select **OK** and wait for the new sync member to be created and deployed.



## 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.

On the **Configure On-Premises** page, do the following things:

1. Select **Choose the Sync Agent Gateway**. The **Select Sync Agent** page opens.

| Select sync members   | Configure On-Premises   | Select Sync Agent   |
|---|---|---|
| Hub Database<br>DataSyncPreviewHubDb  | Choose the Sync Agent Gateway<br><i>Sync Gateway installation is requi...</i> | Existing agents <input checked="" type="radio"/> Create a new agent<br>Download Client Sync Agent<br>It is necessary to install the sync agent client to allow the on premises database connect to your Azure database.<br>Download |
| Member Database<br>Add an Azure Database<br>DataSyncPreviewMember1<br>Add an On-Premises Database | Select the Database<br><i>Not yet selected</i>                                | Agent Name <input type="text"/><br>Create and Generate Key  |
| No On-Premises database available   |   | Generate an agent key<br>Use this key in installed sync agent to register this agent.<br><input type="text"/>   |

2. On the **Choose the Sync Agent Gateway** page, choose whether to use an existing agent or create a new agent.

If you chose **Existing agents**, select the existing agent from the list.

If you chose **Create a new agent**, do the following things:

- a. Download the client sync agent software from the link provided and install it on the computer where the SQL Server is located.

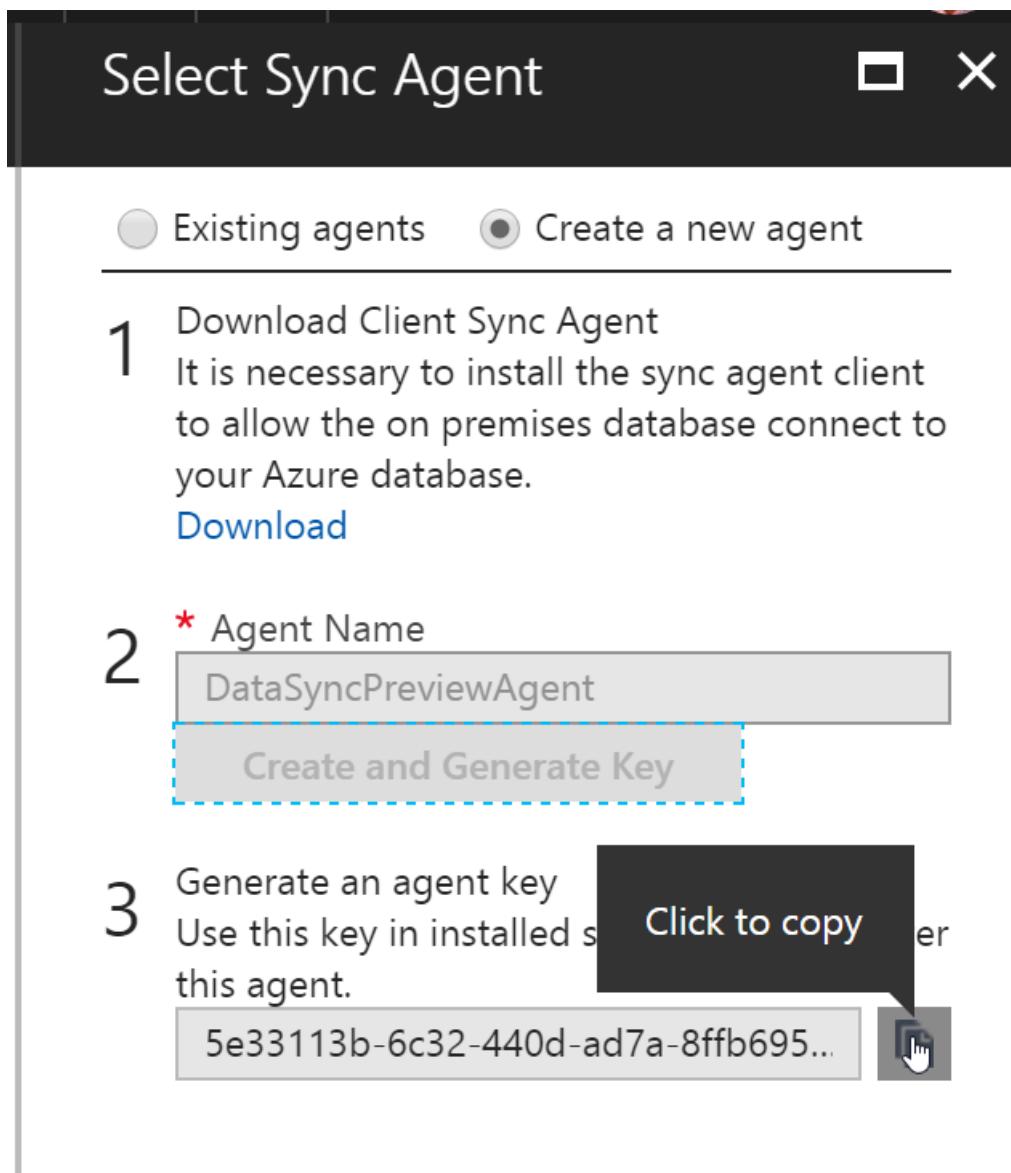
### IMPORTANT

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.

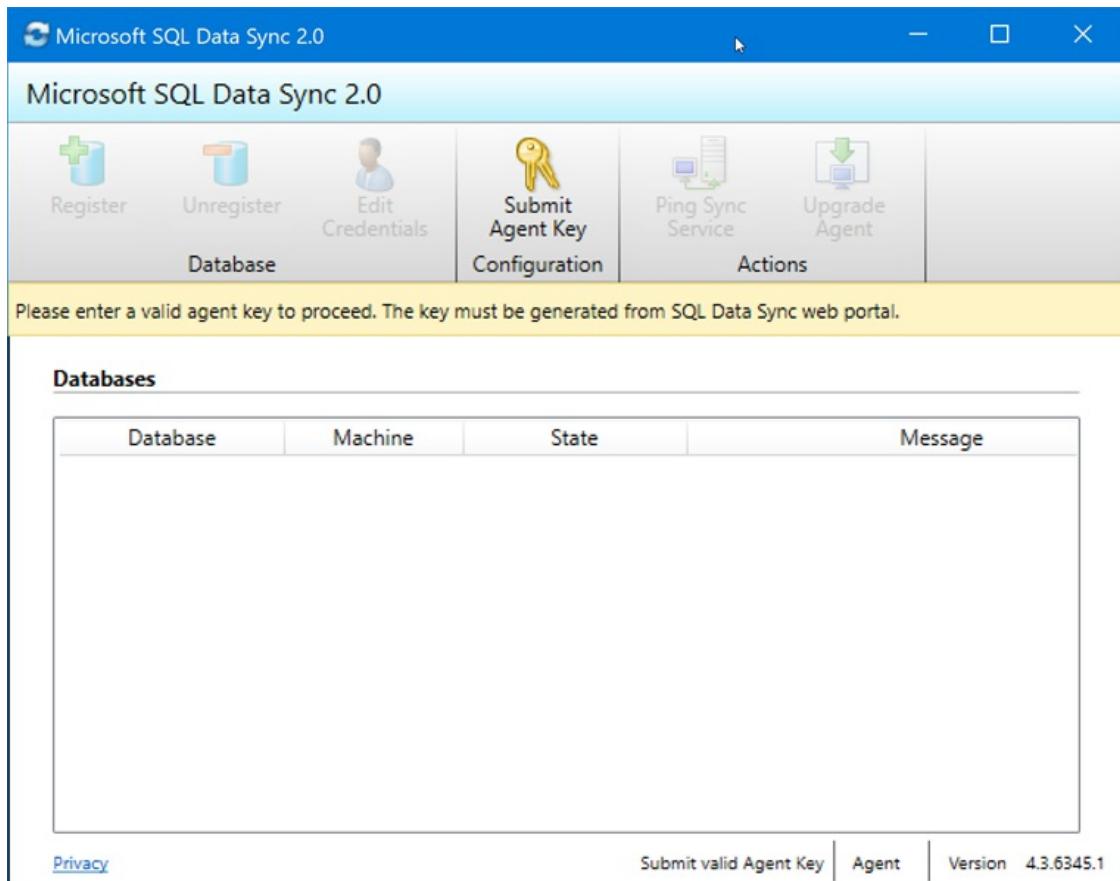
c. Select **Create and Generate Key**.

d. Copy the agent key to the clipboard.

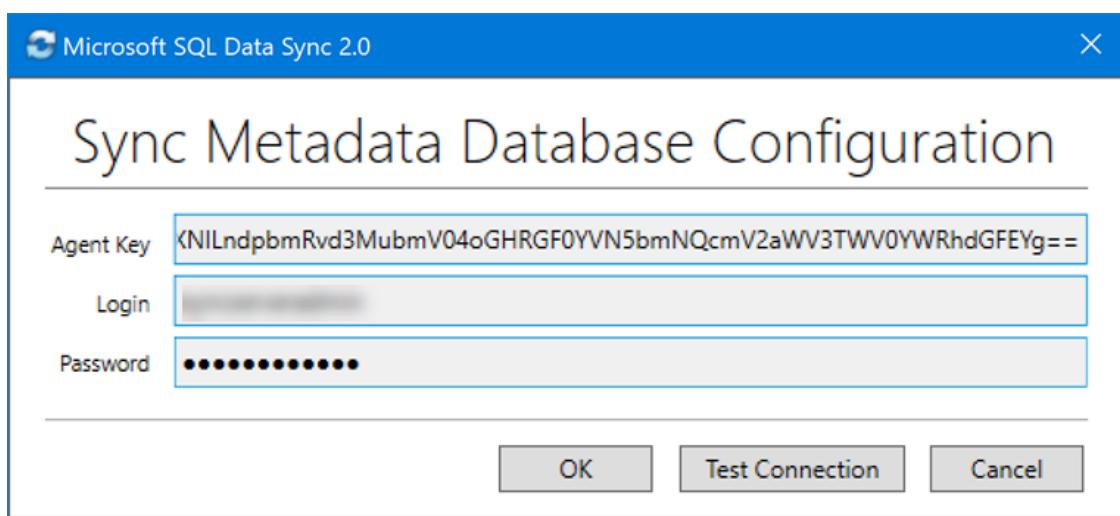


e. Select **OK** to close the **Select Sync Agent** page.

f. On the SQL Server computer, locate and run the Client Sync Agent app.



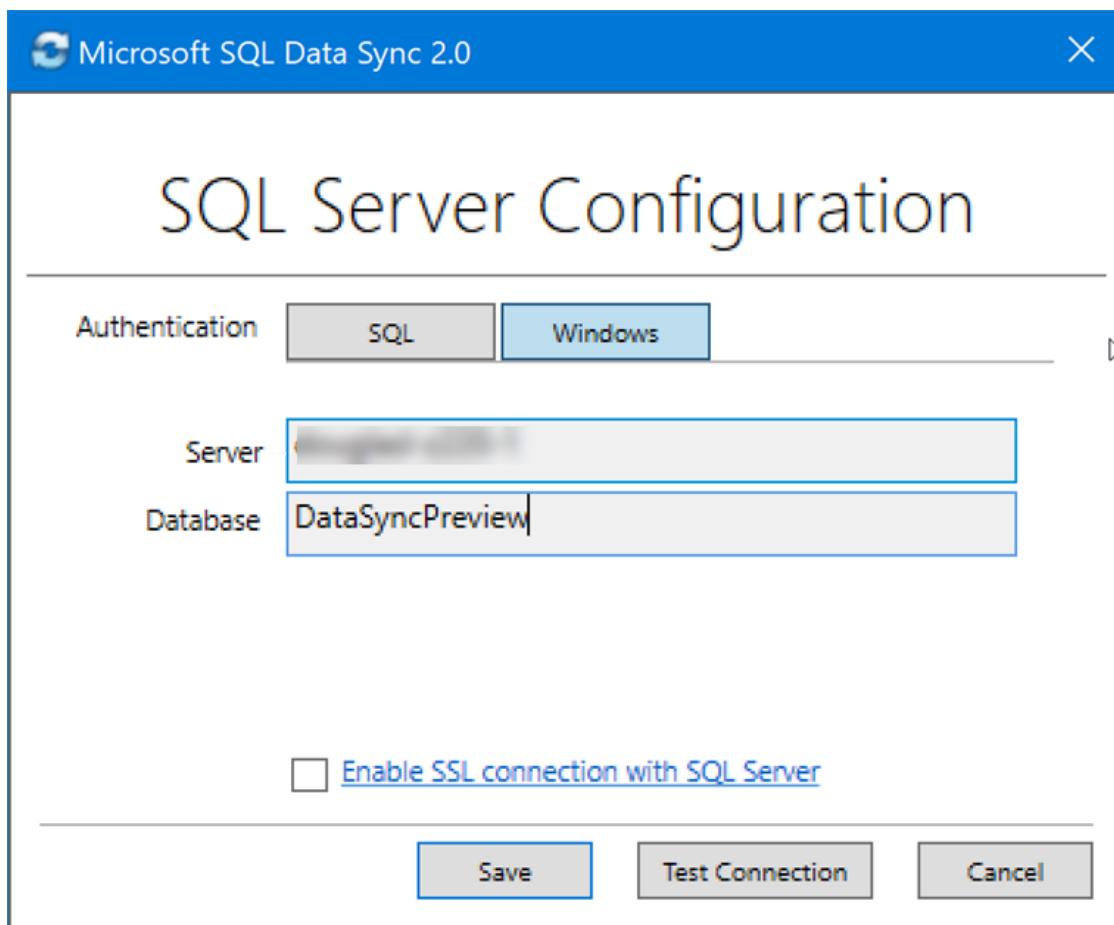
- g. In the sync agent app, select **Submit Agent Key**. The **Sync Metadata Database Configuration** dialog box opens.
- h. In the **Sync Metadata Database Configuration** dialog box, paste in the agent key copied from the Azure portal. Also provide the existing credentials for the Azure SQL Database server on which the metadata database is located. (If you created a new metadata database, this database is on the same server as the hub database.) Select **OK** and wait for the configuration to finish.



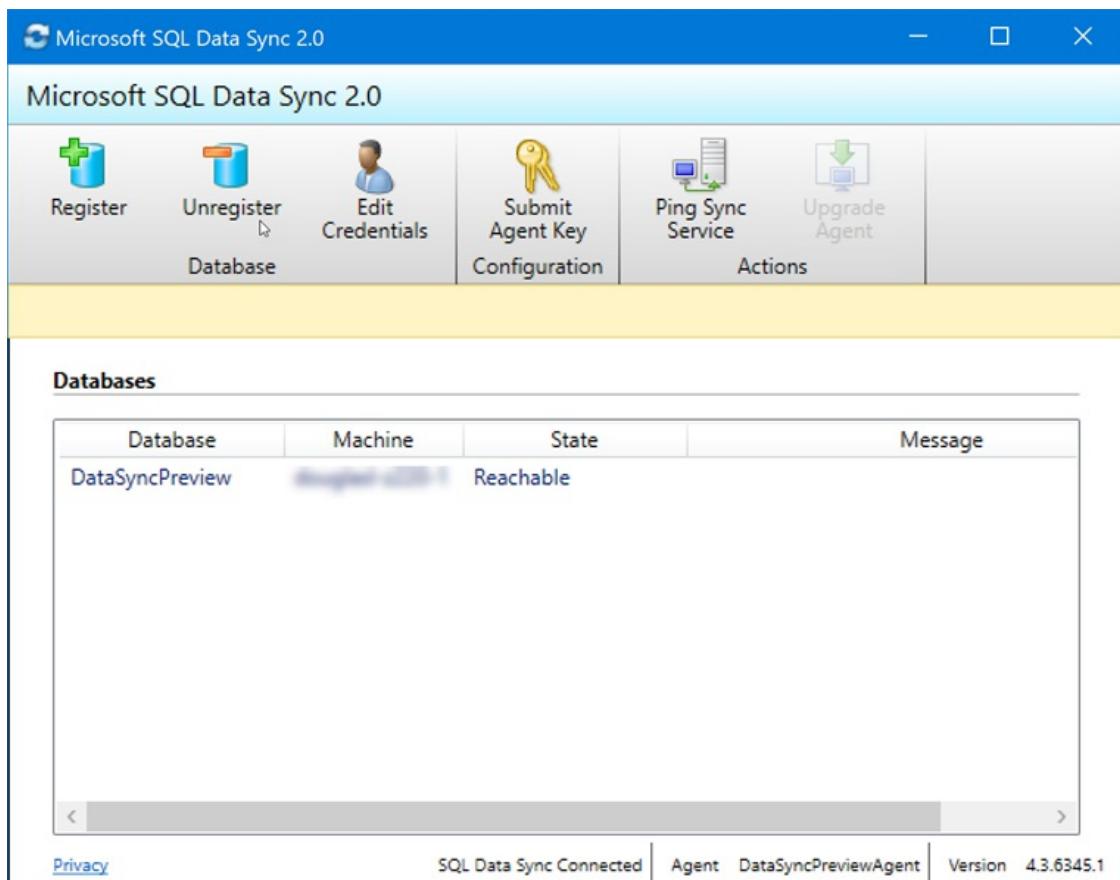
#### NOTE

If you get a firewall error at this point, you have to create a firewall rule on Azure to allow incoming traffic from the SQL Server computer. You can create the rule manually in the portal, but you may find it easier to create it in SQL Server Management Studio (SSMS). In SSMS, try to connect to the hub database on Azure. Enter its name as <hub\_database\_name>.database.windows.net. To configure the Azure firewall rule, follow the steps in the dialog box. Then return to the Client Sync Agent app.

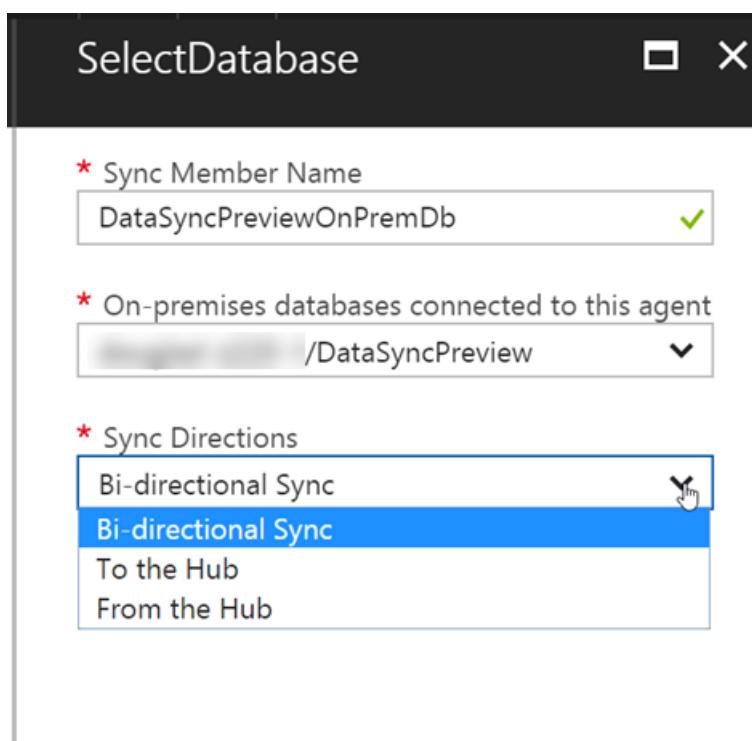
- i. In the Client Sync Agent app, click **Register** to register a SQL Server database with the agent. The **SQL Server Configuration** dialog box opens.



- j. In the **SQL Server Configuration** dialog box, choose whether to connect by using SQL Server authentication or Windows authentication. If you chose SQL Server authentication, enter the existing credentials. Provide the SQL Server name and the name of the database that you want to sync. Select **Test connection** to test your settings. Then select **Save**. The registered database appears in the list.

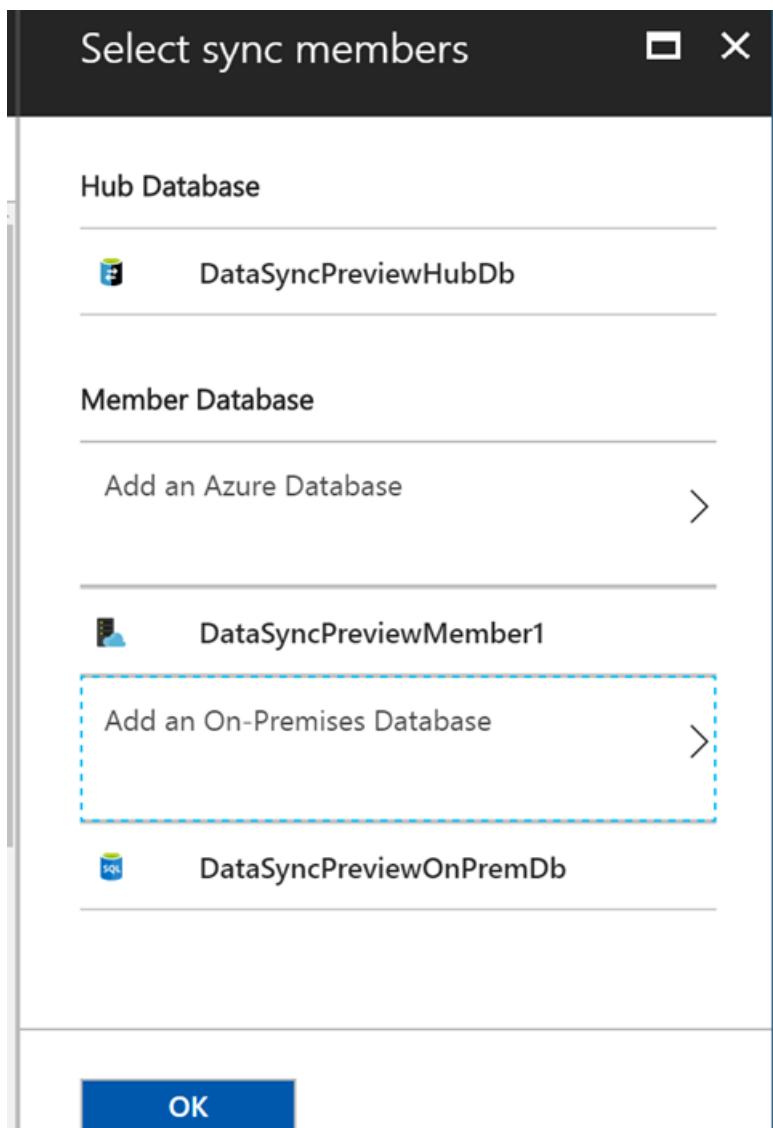


- k. You can now close the Client Sync Agent app.
- l. In the portal, on the **Configure On-Premises** page, select **Select the Database**. The **Select Database** page opens.
- m. On the **Select Database** page, in the **Sync Member Name** field, provide a name for the new sync member. This name is 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.



- n. 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, click **OK** to close the

Select sync members page.



3. 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.

## Step 3 - Configure sync group

After the new sync group members are created and deployed, Step 3, **Configure sync group**, is highlighted in the **New sync group** page.

1. On the **Tables** page, select a database from the list of sync group members, and then select **Refresh schema**.
2. From the list of available tables, select the tables that you want to sync.

The screenshot shows a software interface for managing database tables. At the top, there's a navigation bar with icons for 'Report a bug', 'Search resources', and various system settings. Below the navigation bar, the title 'Tables' is displayed. A dropdown menu labeled 'Select a database' shows 'Hub Database' is selected. A 'Refresh Schema' button is also present.

The main area is divided into two sections:

- Select tables to sync:** This section lists tables from the 'Hub Database'. It includes a column header 'NAME' and a column header 'COLUMNS'. The listed tables are:
  - dbo.BuildVersion (4 columns)
  - dbo.ErrorLog (9 columns)
  - SalesLT.Address (9 columns)
  - SalesLT.Customer (15 columns)
  - SalesLT.CustomerAddr... (5 columns)
  - SalesLT.Product (17 columns)
  - SalesLT.ProductCatego... (5 columns)
  - SalesLT.ProductDescri... (4 columns)
  - SalesLT.ProductModel (5 columns)
  - SalesLT.ProductModel... (5 columns)
  - SalesLT.SalesOrderDet... (9 columns)
  - SalesLT.SalesOrderHea... (22 columns)
- Select fields to sync:** This section has a header row with columns 'NAME', 'DATA TYPE', and 'DESCRIPTIONS'. A large text area below the header says 'Select tables to sync'.

At the bottom left, there is a 'Save' button.

3. By default, all columns in the table are selected. If you don't want to sync all the columns, disable the checkbox for the columns that you don't want to sync. Be sure to leave the primary key column selected.

The screenshot shows the 'Tables' configuration window in the SQL Data Sync interface. At the top, there's a dropdown for 'Select a database' set to 'Hub Database' and a 'Refresh Schema' button. Below this, the left pane lists tables with their column counts: 'dbo.BuildVersion' (4), 'dbo.ErrorLog' (9), 'SalesLT.Address' (9), 'SalesLT.Customer' (15), 'SalesLT.CustomerAddr...' (5), 'SalesLT.Product' (17), 'SalesLT.ProductCatego...' (5), 'SalesLT.ProductDescri...' (4), 'SalesLT.ProductModel' (5), 'SalesLT.ProductModel...' (5), 'SalesLT.SalesOrderDet...' (9), and 'SalesLT.SalesOrderHea...' (22). The 'SalesLT.Product' table is currently selected. The right pane shows the 'Select fields to sync' section for this table, listing columns: ProductID (int(4)), Name (userdefineddatatype(50)), ProductNumber (nvarchar(25)), Color (nvarchar(15)), StandardCost (money(8)), ListPrice (money(8)), Size (nvarchar(5)), Weight (decimal(5)), ProductCategoryID (int(4)), ProductModelID (int(4)), SellStartDate (datetime(8)), SellEndDate (datetime(8)), DiscontinuedDate (datetime(8)), ThumbNailPhoto (varbinary(max)), ThumbnailPhotoFile... (nvarchar(50)), rowguid (uniqueidentifier(16)), and ModifiedDate (datetime(8)). Most columns have a checked checkbox next to them. A 'Save' button is located at the bottom of the right pane.

4. Finally, select **Save**.

## FAQ about setup and configuration

### How frequently can Data Sync synchronize my data

The minimum frequency is every five minutes.

### Does SQL Data Sync fully create and provision tables

If the sync schema tables are not already created in the destination database, SQL Data Sync creates them with the columns that you selected. However, this behavior does not result in a full fidelity schema, for the following reasons:

- Only the columns that you selected are created in the destination table. If some columns in the source tables are not part of the sync group, those columns are not provisioned in the destination tables.
- Indexes are created only for the selected columns. If the source table index has columns that are not part of the sync group, those indexes are not provisioned in the destination tables.
- Indexes on XML type columns are not provisioned.
- CHECK constraints are not provisioned.
- Existing triggers on the source tables are not provisioned.
- Views and stored procedures are not created on the destination database.

Because of these limitations, we recommend the following things:

- For production environments, provision the full-fidelity schema yourself.
- For trying out the service, the auto-provisioning feature of SQL Data Sync works well.

### Why do I see tables that I did not create

Data Sync creates side tables in your database for change tracking. Don't delete them or Data Sync stops working.

### Is my data convergent after a sync

Not necessarily. In a sync group with a hub and three spokes (A, B, and C), the 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 is not written to either database B or database C until the next sync task.

### How do I get schema changes into a sync group

You have to 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.

### Adding new tables and columns.

New tables and columns don't impact the current sync. Data Sync ignores the new tables and columns until you add them to the sync schema. When you add new database objects, this is the best sequence to follow:

1. Add the new tables or columns to the hub and to all sync members.
2. Add the new tables or columns to the sync schema.
3. Start to insert values into the new tables and columns.

### 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 that's too large for the **int** data type. To complete the change, replicate the schema change manually to the hub and to all sync members, and 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 new database, you have to do the following two things to use Data Sync in the new database:

1. Clean up the Data Sync objects and side tables on the **new database** by using [this script](#). This script deletes all of 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.

## FAQ about the client agent

### 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.

## Next steps

Congratulations. You have created a sync group that includes both a SQL Database instance and a SQL Server database.

For more info about SQL Data Sync, see:

- [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- [Best practices for Azure SQL Data Sync](#)
- [Monitor Azure SQL Data Sync with Log Analytics](#)
- [Troubleshoot issues with Azure SQL Data Sync](#)
- Complete PowerShell examples that show how to configure SQL Data Sync:
  - [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](#)
- [Download the SQL Data Sync REST API documentation](#)

For more info about SQL Database, see:

- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Azure CLI samples for Azure SQL Database

9/24/2018 • 2 minutes to read • [Edit Online](#)

The following table includes links to Azure CLI script examples for Azure SQL Database.

| <b>Create a single database and an elastic pool</b>                    |  |
|--|--|
| <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.                                     |
| <a href="#">Create elastic pools and move pooled databases</a>         | 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>                     |  |
| <a href="#">Scale a single database</a>                                | This CLI script example scales a single Azure SQL database to a different compute size after querying the size information for the database. |
| <a href="#">Scale an elastic pool</a>                                  | This CLI script example scales a SQL elastic pool to a different compute size.   |
| <a href="#">Create and manage a Managed Instance</a>                   | These CLI scripts show you have to create and manage a Managed Instance using the Azure CLI  |
|  |  |

# Azure PowerShell samples for Azure SQL Database

10/29/2018 • 2 minutes to read • [Edit Online](#)

The following table includes links to sample Azure PowerShell scripts for Azure SQL Database.

| <b>Create a single database and an elastic pool</b>                                   |   |
|---|---|
| <a href="#">Create a single database and configure a 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.   |
| <a href="#">Create and manage a Managed Instance</a>                                  | This PowerShell script shows you how to create and manage a Managed Instance using the Azure PowerShell   |
| <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.   |
| <a href="#">Configure and failover a failover group for a single database</a>         | This PowerShell script configures a failover group for an Azure SQL Database server instance, adds a database to the failover group, and fails it over to the secondary server                  |
| <b>Scale a single databases 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.   |
| <a href="#">Restore, copy, and import a database</a>                                  |   |

|   |   |
|---|---|
| <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.  |

# Azure SQL Database purchasing models

10/19/2018 • 7 minutes to read • [Edit Online](#)

Azure SQL Database enables you to easily purchase fully managed PaaS database engine that fits your performance and cost needs. Depending on the deployment model of Azure SQL Database, you can select the purchasing model that fits your needs:

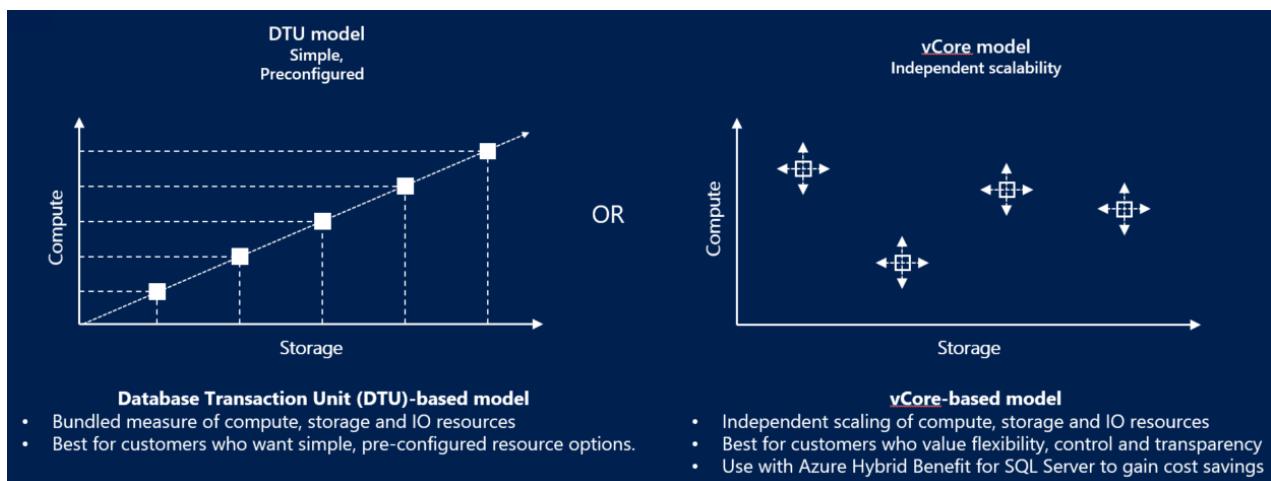
- [Logical servers in Azure SQL Database](#) offers two purchasing models for compute, storage, and IO resources: a [DTU-based purchasing model](#) and a [vCore-based purchasing model](#). Within this purchasing model, you can choose [single databases](#) or [elastic pools](#).
- [Managed Instances](#) in Azure SQL Database only offer the [vCore-based purchasing model](#).

## IMPORTANT

[Hyperscale databases \(preview\)](#) are in public preview only for single databases using the vCore purchasing model.

The following table and chart compare and contrast these two purchasing models.

| PURCHASING MODEL  | DESCRIPTION   | BEST FOR   |
|-------------------|---|--|
| DTU-based model   | This model is based on a bundled measure of compute, storage, and IO resources. 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 <a href="#">What are DTUs and eDTUs?</a> | Best for customers who want simple, pre-configured resource options. |
| vCore-based model | This model allows you to independently choose compute and storage resources. It also allows you to use Azure Hybrid Benefit for SQL Server to gain cost savings.  | Best for customers who value flexibility, control, and transparency. |



## vCore-based purchasing model

A virtual core represents the logical CPU offered with an option to choose between generations of hardware and physical characteristics of hardware (for example, number of cores, memory, storage size). The vCore-based purchasing model gives your 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 based upon their workload needs. In the vCore-based purchasing model, you can choose between [General Purpose](#) and [Business critical](#) service tiers for both [single databases](#), [managed instances](#), and [elastic pools](#). For single databases, you can also choose the [Hyperscale \(preview\)](#) service tier.

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

- Compute (service tier + number of vCores and amount of memory + generation of hardware)
- Type and amount of data and log storage
- Backup storage (RA-GRS)

#### IMPORTANT

Compute, IOs, data and log storage are charged per database or elastic pool. Backups storage is charged per each database. For details of Managed Instance charges, refer to [Azure SQL Database Managed Instance](#). **Region limitations:** The vCore-based purchasing model is not yet available in the following regions: West Europe, France Central, UK South, UK West and Australia Southeast.

If your database or elastic pool consumes more than 300 DTU conversion to vCore may reduce your cost. You can convert using your API of choice or using the Azure portal, with no downtime. However, conversion is not required. If the DTU purchasing model meets your performance and business requirements, you should continue using it. If you decide to convert from the DTU-model to vCore-model, you should select the compute size using the following rule of thumb: each 100 DTU in Standard tier requires at least 1 vCore in General Purpose tier; each 125 DTU in Premium tier requires at least 1 vCore in Business Critical tier.

## DTU-based purchasing model

The 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. Customers who prefer the simplicity of a pre-configured bundle and fixed payments each month, may find the DTU-based model more suitable for their needs. In the DTU-based purchasing model, customers can choose between **Basic**, **Standard**, and **Premium** service tiers for both [single databases](#) and [elastic pools](#). This purchase model is not available in [managed instances](#).

### Database Transaction Units (DTUs)

For a single Azure SQL 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), providing a predictable level of performance. The amount of resources is calculated as a number of Database Transaction Units or DTUs and is a bundled measure of compute, storage, and IO resources. The ratio amongst these resources was originally determined by an [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 timeouts. The resources used by your workload do not impact the resources available to other SQL databases in the Azure cloud, and the resources used by other workloads do not impact the resources available to your SQL database.

# Database Transaction Unit – DTU

## Bounding box

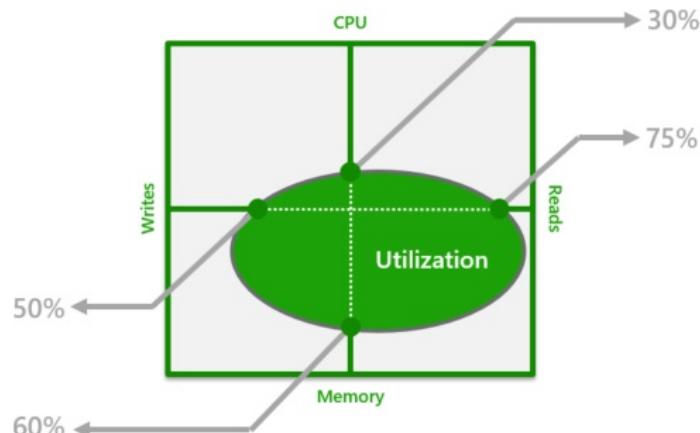
Monitoring database workload utilization within bounding box

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 amount of resources between 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. For example, a Premium P11 database with 1750 DTUs provides 350x more DTU compute power than a Basic database with 5 DTUs.

To gain deeper insight into the resource (DTU) consumption of your workload, use [Azure SQL Database Query Performance Insight](#) to:

- Identify the top queries by CPU/Duration/Execution count that can potentially be tuned for improved performance. For example, an IO intensive query might benefit from the use of [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, view its text and history of resource utilization.
- Access performance tuning recommendations that show actions performed by [SQL Database Advisor](#).

## Elastic Database Transaction Units (eDTUs)

Rather than provide a dedicated set of resources (DTUs) that may not always be needed for a SQL Database that is always available, you can place databases into an [elastic pool](#) on a SQL Database server that shares a pool of resources among those databases. The shared resources in an elastic pool are measured by elastic Database Transaction Units or eDTUs. Elastic pools provide a simple cost effective solution to manage the performance goals for multiple databases having widely varying and unpredictable usage patterns. An elastic pool guarantees resources cannot be consumed by one database in the pool, while ensuring 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. Within the elastic pool, individual databases are given the flexibility to auto-scale within the configured boundaries. A database under heavier load will consume more eDTUs to meet demand. Databases under lighter loads will consume less eDTUs. Databases with no load will consume no eDTUs. By provisioning resources for the entire pool, rather than per database, management tasks are simplified, providing a predictable budget for the pool.

Additional eDTUs can be added to an existing pool with no database downtime and with no impact on the databases in the pool. Similarly, if extra eDTUs are no longer needed, they can be removed from an existing pool at any point in time. You can add or subtract databases to the pool or limit the amount of eDTUs a database can use under heavy load to reserve eDTUs for other databases. If a database is predictably under-utilizing resources, you can 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 are looking to migrate an existing on-premises or SQL Server virtual machine workload to Azure SQL Database, you can use the [DTU Calculator](#) to approximate the number of DTUs needed. For an existing Azure SQL

Database workload, you can use [SQL Database Query Performance Insight](#) to understand your database resource consumption (DTUs) to gain deeper insight for optimizing your workload. You can also use the [sys.dm\\_db\\_resource\\_stats](#) DMV to view resource consumption for the last hour. Alternatively, the catalog view [sys.resource\\_stats](#) displays resource consumption for the last 14 days, but at a lower fidelity of five-minute averages.

### **Workloads that benefit from an elastic pool of resources**

Pools are suited for a large number of databases with specific utilization patterns. For a given database, this pattern is characterized by a low utilization average with relatively infrequent utilization spikes. SQL Database automatically evaluates the historical resource usage of databases in an existing SQL Database server and recommends the appropriate pool configuration in the Azure portal. For more information, see [when should an elastic pool be used?](#)

## Next steps

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

# vCore service tiers, Azure Hybrid Benefit, and migration

10/23/2018 • 6 minutes to read • [Edit Online](#)

The vCore-based purchasing model enables you to independently scale compute and storage resources, match on-premises performance, and optimize price. It also enables you to choose generation of hardware:

- Gen 4 - Up to 24 logical CPUs based on Intel E5-2673 v3 (Haswell) 2.4 GHz processors, vCore = 1 PP (physical core), 7 GB per core, attached SSD
- Gen 5 - Up to 80 logical CPUs based on Intel E5-2673 v4 (Broadwell) 2.3 GHz processors, vCore=1 LP (hyper-thread), 5.5. GB per core, fast eNVM SSD

vCore model also allows you to use [Azure Hybrid Benefit for SQL Server](#) to gain cost savings.

## NOTE

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

## Service tier characteristics

The vCore model provides two service tiers General Purpose and Business Critical. Service tiers are differentiated by a range of compute sizes, high availability design, fault isolation, types of storage and IO range. The customer must separately configure the required storage and retention period for backups. You must separately configure the required storage and retention period for backups. In the Azure portal, go to Server (not the database) > Managed Backups > Configure Policy > Point In Time Restore Configuration > 7 - 35 days.

The following table helps you understand the differences between these two tiers:

|          | GENERAL PURPOSE  | BUSINESS CRITICAL   | HYPERSCALE (PREVIEW)   |
|----------|--|---|--|
| Best for | Most business workloads. Offers budget oriented balanced and scalable compute and storage options. | Business applications with high IO requirements. Offers highest resilience to failures using several isolated replicas. | Most business workloads with highly scalable storage and read-scale requirements |
| Compute  | Gen4: 1 to 24 vCore<br>Gen5: 1 to 80 vCore   | Gen4: 1 to 24 vCore<br>Gen5: 1 to 80 vCore  | Gen4: 1 to 24 vCore<br>Gen5: 1 to 80 vCore                                       |
| Memory   | Gen4: 7 GB per core<br>Gen5: 5.5 GB per core   | Gen4: 7 GB per core<br>Gen5: 5.5 GB per core  | Gen4: 7 GB per core<br>Gen5: 5.5 GB per core                                     |

|                             | GENERAL PURPOSE  | BUSINESS CRITICAL   | HYPERSCALE (PREVIEW)   |
|-----------------------------|--|---|--|
| Storage                     | Premium remote storage, Single database: 5 GB – 4 TB<br>Managed Instance: 32 GB - 8 TB                                     | Local SSD storage, Single database: 5 GB – 4 TB<br>Managed Instance: 32 GB - 4 TB | Flexible, autogrow of storage as needed. Supports up to 100 TB storage and beyond. Local SSD storage for local buffer pool cache and local data storage. Azure remote storage as final long-term data store.   |
| IO throughput (approximate) | Single database: 500 IOPS per vCore with 7000 maximum IOPS<br>Managed Instance:<br>Depends on <a href="#">size of file</a> | 5000 IOPS per core with 200,000 maximum IOPS                                      | TBD  |
| Availability                | 1 replica, no read-scale   | 3 replicas, 1 <a href="#">read-scale replica</a> , zone redundant HA              | ?  |
| Backups                     | <a href="#">RA-GRS</a> , 7-35 days (7 days by default)   | <a href="#">RA-GRS</a> , 7-35 days (7 days by default)                            | snapshot-based backup in Azure remote storage and restores use these snapshots for fast recovery. Backups are instantaneous and do not impact the IO performance of Compute. Restores are very fast and are not a size of data operation (taking minutes rather than hours or days). |
| In-Memory                   | Not supported  | Supported   | Not supported  |

#### NOTE

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

- For more information, see [vCore resource limits in Single database](#) and [vCore resource limits in Managed Instance](#).
- For more information about the General Purpose and Business Critical service tiers, see [General Purpose and Business Critical service tiers](#).
- For details on the Hyperscale service tier in the vCore-based purchasing model, see [Hyperscale service tier](#).

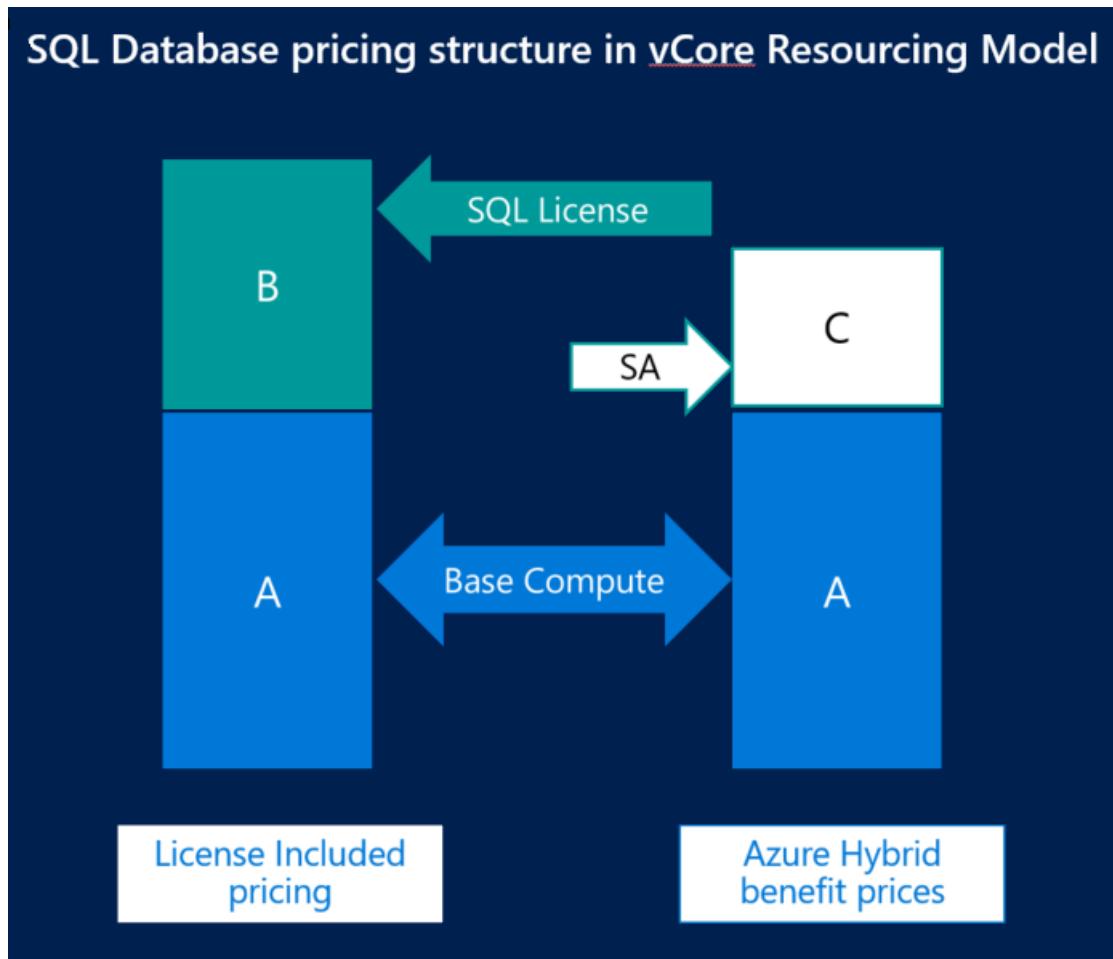
#### IMPORTANT

If you need less than one vCore of compute capacity, use the DTU-based purchasing model.

See [SQL Database FAQ](#) for answers to frequently asked questions.

## Azure Hybrid Benefit

In the vCore-based purchasing model, you can exchange your existing licenses for discounted rates on SQL Database using the [Azure Hybrid Benefit for SQL Server](#). This Azure benefit allows you to use your on-premises SQL Server licenses to save up to 30% on Azure SQL Database using your on-premises SQL Server licenses with Software Assurance.



## Migration from DTU model to vCore model

### Migration of a database

Migrating a database from the DTU-based purchasing model to the vCore-based purchasing model is similar to upgrading or downgrading between Standard and Premium databases in the DTU-based purchasing model.

### Migration of databases with geo-replication links

Migrating to from DTU-based model to vCore-based model is similar to upgrading or downgrading the geo-replication relationships between Standard and Premium databases. It does not require terminating geo-replication but the user must observe the 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 using geo-replication between two elastic pools, it is recommended that you designate one pool as the primary and the other – as the secondary. In that case, migrating elastic pools should use the same guidance. However, it is technically it is possible that an elastic pool contains both primary and secondary databases. In this case, to properly migrate you should treat the pool with the higher utilization as “primary” and follow the sequencing rules accordingly.

The following table provides guidance for the 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 an 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  |

\* Each 100 DTU in Standard tier requires at least 1 vCore and each 125 DTU in Premium tier requires at least 1 vCore

### Migration of 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 model, the failover group will remain in effect with the same policy settings.

### Creation of a geo-replication secondary

You can only create a geo-secondary using the same service tier as the primary. For database with high log generation rate, it is advised that the secondary is created with the same compute size as the primary. If you are creating a geo-secondary in the elastic pool for a single primary database, it is advised that the pool has the `maxVCore` setting that matches the primary database compute size. If you are creating a geo-secondary in the elastic pool for a primary in another elastic pool, it is advised that the pools have the same `maxVCore` settings

### Using 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 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 database, 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](#)

Database vCore-based resource limits for elastic pools.

# General Purpose and Business Critical service tiers

10/16/2018 • 2 minutes to read • [Edit Online](#)

This article discusses storage and backup considerations for the General Purpose and Business Critical service tiers in the vCore-based purchasing model.

## NOTE

For details on 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

Consider the following:

- The allocated storage is used by data files (MDF) and log files (LDF) files.
- Each single database compute size supports a maximum database size, with a default max size of 32 GB.
- When you configure the required single database size (size of MDF), 30% of additional storage is automatically added to support LDF
- Storage size in 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 Standard storage, increase or decrease size in 10-GB increments
  - For Premium storage, increase or decrease 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.

To monitor the current total size of MDF and LDF, 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 [RA-GRS storage](#) are automatically. The storage size increases

dynamically as the new backups are created. The storage is used by weekly full backups, daily differential backups, and transaction log backups copied every 5 minutes. The storage consumption depends on the rate of change of the database and the retention period. You can configure a separate retention period for each database between 7 and 35 days. A minimum storage amount equal to 1x of data size is provided at no extra charge. For most databases, this amount is enough to store 7 days of backups.

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

## Next steps

- For details on specific compute sizes and storage size choices available for single database in the General Purpose and Business Critical Service tiers, see [SQL Database vCore-based resource limits for single databases](#)
- For details on specific compute sizes and storage size choices available for elastic pools in the General Purpose and Business Critical Service tiers, see [SQL Database vCore-based resource limits for elastic pools](#).

# Hyperscale service tier (preview) for up to 100 TB

10/17/2018 • 10 minutes to read • [Edit Online](#)

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.

## IMPORTANT

Hyperscale service tier is currently in public preview and available in limited Azure regions. For the full region list, see [Hyperscale service tier available regions](#). We don't recommend running any production workload in Hyperscale databases yet. You can't update a Hyperscale database to other service tiers. For test purpose, we recommend you make a copy of your current database, and update the copy to Hyperscale service tier.

## 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](#).

## IMPORTANT

Hyperscale service tier is currently in public preview. We don't recommend running any production workload in Hyperscale databases yet. You can't update a Hyperscale database to other service tiers. For test purpose, we recommend you make a copy of your current database, and update the copy to Hyperscale service tier.

## 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
- 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 primarily intended for customers who have large databases either on-premises and want to modernize their applications by moving to the cloud or for customers who are already in the cloud and are limited by the maximum database size restrictions (1-4 TB). It is also intended for customers who seek high performance and high scalability for storage and compute.

The Hyperscale service tier supports all SQL Server workloads, but it is primarily optimized for OLTP. The Hyperscale service tier also supports hybrid and analytical (data mart) 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. In public preview, we create two replicas per Hyperscale database by default.

- **Storage:**

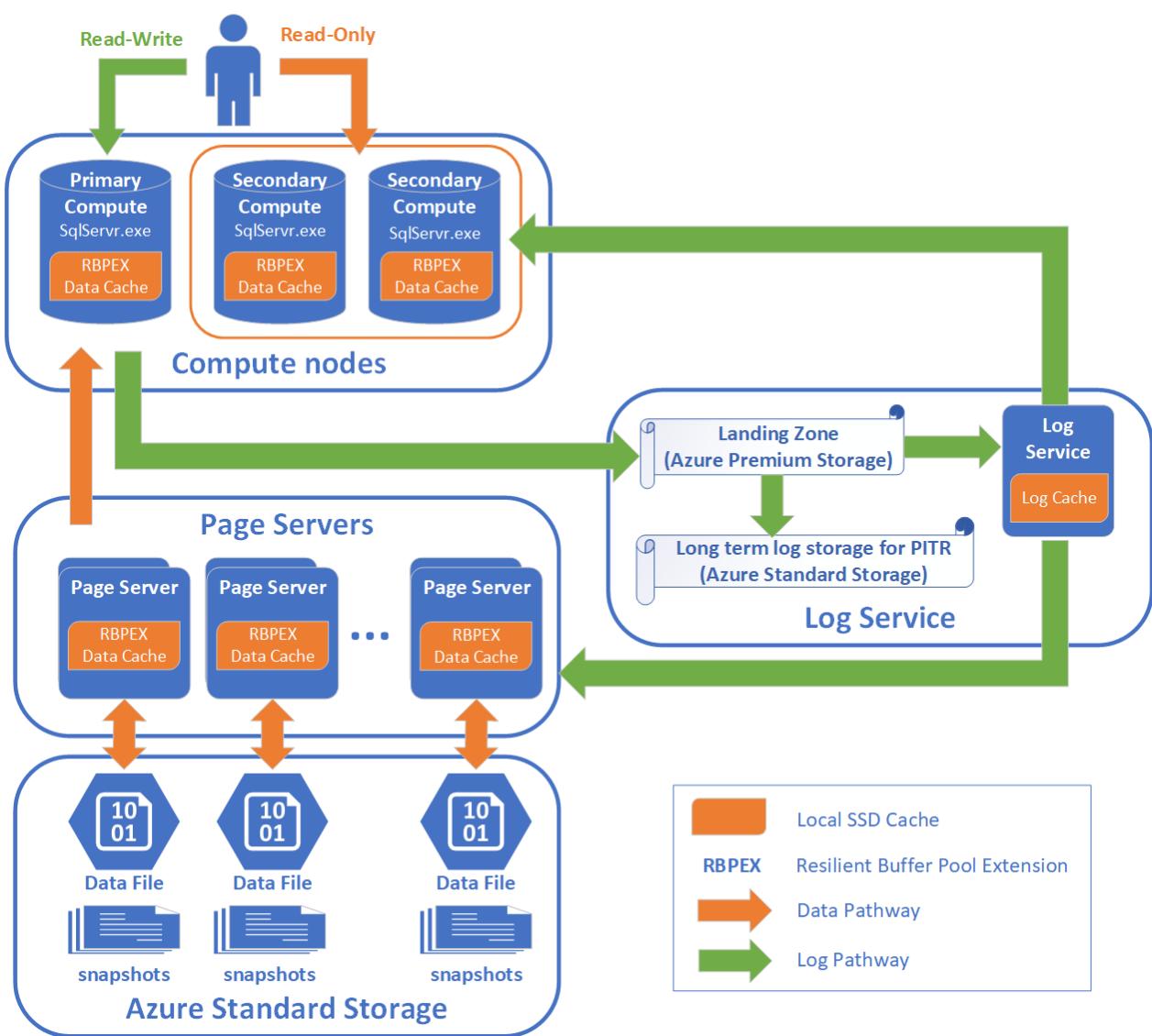
You don't need to specify the max data size when configuring a Hyperscale database. In the hyperscale tier, you are charged for storage for your database based on actual usage. Storage is dynamically allocated between 5 GB and 100 TB, in 1 GB increments.

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 compute components so no data copy is required to spin up a new readable replica. During preview, only 1 read-only replica is supported.

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



A Hyperscale database contains the following different types of nodes:

### Compute node

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 node

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 1 terabyte 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 node

The log service node accepts log records from the primary compute node, persists them in a durable cache, and forwards the log records to the rest of the compute nodes (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 node are propagated through the log service to all the secondary compute nodes and page servers. Finally, the log record(s) are pushed out to long-term storage in Azure Storage, which is an infinite storage repository. This mechanism removes the necessity for frequent log truncation. The log service also has local cache to speed up access.

### Azure storage node

The Azure storage node is the final destination of data from page servers. This storage is used for backup purposes as well as for replication between Azure regions. Backups consist of snapshots of data files. Restore operation are fast from these snapshots and data can be restored to any point in time.

## Backup and restore

Backups are file-snapshot base and hence they are nearly instantaneous. Storage and compute separation enable pushing down the backup/restore operation to the storage layer to reduce the processing burden on the primary compute node. As a result, the backup of a large database does not impact the performance of the primary compute node. Similarly, restores are done by copying the file snapshot and as such are not a size of data operation. For restores within the same storage account, the restore operation is fast.

## 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.

```
-- Create a HyperScale Database
CREATE DATABASE [HyperScaleDB1] (EDITION = 'HyperScale', SERVICE_OBJECTIVE = 'HS_Gen4_4');
GO
```

## 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](#). In public preview, this is a one-way migration. You can't move databases from Hyperscale to another service tier. We recommend you make a copy of your production databases and migrate to Hyperscale for proof of concepts (POCs).

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_Gen4_4');
GO
```

## IMPORTANT

Transparent Database Encryption (TDE) should be turned off before altering a non-Hyperscale database to HyperScale.

## Connect to a read-scale replica of a Hyperscale database

In HyperScale 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;
```

## Available regions

Hyperscale service tier is currently in public preview and available in following Azure regions: EastUS1, EastUS2, WestUS2, CentralUS, NorthCentralUS, WestEurope, NorthEurope, UKWest, AustraliaEast, AustraliaSouthEast, SouthEastAsia, JapanEast, KoreaCentral

## Known limitations

| 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.   |
| If a database file grows during migration due to an active workload and crosses the 1 TB per file boundary, the migration fails | Mitigations:<br>- If possible, migrate the database when there is no update workload running.<br>- Re-try the migration, it will succeed as long as the 1 TB boundary is not crossed during the migration.  |
| Managed Instance is not currently supported   | Not currently supported   |
| 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.             |
| Migration of databases with in-memory objects is not currently supported  | In-Memory objects must be dropped and recreated as non-In-Memory objects before migrating a database to the Hyperscale service tier.  |

## 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](#).

# FAQ about Azure SQL Hyperscale databases

10/17/2018 • 17 minutes to read • [Edit Online](#)

This article provides answers to frequently asked questions for customers considering a database in the Azure SQL Database Hyperscale service tier, commonly called a Hyperscale database (currently in public preview). This article describes the scenarios that Hyperscale supports and the cross-feature services are compatible with SQL Database Hyperscale in general.

- This FAQ is intended for readers who have a brief understanding of the Hyperscale 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 SQL Database Hyperscale database. For that, we recommend you refer to the [Azure SQL Database Hyperscale](#) documentation.

## General questions

### What is a Hyperscale 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, and so on, work like any other 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 primarily differentiated based upon availability, storage type and IOPs.

- The General Purpose service tier is appropriate for most business workloads, offering a balanced set of compute and storage options where IO latency or failover times are not the priority.
- The Hhyperscale service tier is optimized for very large database workloads.
- The Business Critical service tier is appropriate for business workloads where IO latency is a priority.

|                      | RESOURCE TYPE                    | GENERAL PURPOSE   | HYPERSCALE   | BUSINESS CRITICAL   |
|----------------------|----------------------------------|---|--|---|
| <b>Best for</b>      | All                              | Most business workloads. Offers budget oriented balanced compute and storage options. | Data applications with large data capacity requirements and the ability to auto-scale storage and scale compute fluidly. | OLTP applications with high transaction rate and lowest latency IO. Offers highest resilience to failures using several, isolated replicas. |
| <b>Resource type</b> |                                  | Single database / elastic pool / managed instance                                     | Single database  | Single database / elastic pool / managed instance   |
| <b>Compute size</b>  | Single database / elastic pool * | 1 to 80 vCores  | 1 to 80 vCores*  | 1 to 80 vCores  |

|                      | <b>RESOURCE TYPE</b>           | <b>GENERAL PURPOSE</b>                    | <b>HYPERSCALE</b>  | <b>BUSINESS CRITICAL</b>                                      |
|----------------------|--------------------------------|---|--|---|
|                      | Managed instance               | 8, 16, 24, 32, 40, 64, 80 vCores          | N/A  | 8, 16, 24, 32, 40, 64, 80 vCores                              |
| <b>Storage type</b>  | All                            | Premium remote storage (per instance)     | De-coupled storage with local SSD cache (per instance)                             | Super-fast local SSD storage (per instance)                   |
| <b>Storage size</b>  | 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  |
| <b>IO throughput</b> | Single database**              | 500 IOPS per vCore with 7000 maximum IOPS | Unknown yet  | 5000 IOPS with 200,000 maximum IOPS                           |
|                      | Managed instance               | Depends on size of file                   | N/A  | Managed Instance: Depends on size of file                     |
| <b>Availability</b>  | All                            | 1 replica, no read-scale, no local cache  | Multiple replicas, up to 15 read-scale, partial local cache                        | 3 replicas, 1 read-scale, zone-redundant HA, full local cache |
| <b>Backups</b>       | All                            | RA-GRS, 7-35 days (7 days by default)     | RA-GRS, 7-35 days (7 days by default), constant time point-in-time recovery (PITR) | RA-GRS, 7-35 days (7 days by default)                         |

\* Elastic pools not supported in the Hyperscale service tier

### Who should use the Hyperscale service tier

The Hyperscale service tier is primarily 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:

- Support for up to 100 TB of database size
- Fast database backups regardless of database size (backups are based on file snapshots)
- Fast database restores regardless of database size (restores are from file snapshots)
- Higher log throughput results in fast transaction commit times regardless of database size
- Read scale out to one or more read-only nodes to offload your read workload, and for 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 to a P11, for example, but much faster as this is not a size of data operation.

### What regions currently support Hyperscale

Hyperscale is currently available for single databases in the following regions: West US1, West US2, East US1, Central US, West Europe, North Europe, UK West, SouthEast Asia, Japan East, Korea Central, Australia SouthEast, and Australia East.

### 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 characteristic of a Hyperscale database

The SQL Database Hyperscale architecture provides high performance and throughput while supporting large database sizes. The precise performance profile and characteristics is not available during public preview.

## What is the scalability of a Hyperscale database

SQL Database Hyperscale provides rapid scalability based on your workload demand.

- **Scaling Up/Down**

With Hyperscale, you can scale up the primary compute size in terms of resources like CPU, 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 Hyperscale, you also get the ability to provision one or more additional compute nodes that you can use to serve your read requests. This means that you can use these additional compute nodes as read-only nodes to offload your read workload from the primary compute. In addition to read-only, these nodes also serve as hot-standby's in the event of a fail over from the primary.

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

## Deep dive questions

### Can I mix Hyperscale and single databases a my logical server

Yes, you can.

### Does Hyperscale 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 modes to interact with your Azure SQL database.

### What transaction isolation levels are going to be default on SQL Database Hyperscale database

On the primary node, the transaction isolation level is RCSI (Read Committed Snapshot Isolation). On the read scale secondary nodes, the isolation level is Snapshot.

### Can I bring my on-premises or IaaS SQL Server license to SQL Database Hyperscale

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

### What kind of workloads is SQL Database Hyperscale designed for

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

### How can I choose between Azure SQL Data Warehouse and SQL Database Hyperscale

If you are currently running interactive analytics queries using SQL Server as a data warehouse, SQL Database Hyperscale is a great option because you can host relatively small data warehouses (such as a few TB up to 10's of TB) at a lower cost and you can migrate your data warehouse workload to SQL Database Hyperscale without T-SQL code changes.

If you are running data analytics on a large scale with complex queries and using Parallel Data Warehouse (PDW), Teradata or other Massively Parallel Processor (MPP) data warehouses, SQL Data Warehouse may be the best choice.

## SQL Database Hyperscale compute questions

### Can I pause my compute at any time

No.

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

No. To get more RAM, you need to upgrade to a higher compute size. Gen4 hardware provides more RAM compared to Gen5 hardware. For more information, see [Hyperscale storage and compute sizes](#).

### Can I provision multiple compute nodes of different sizes

No.

### How many read-scale replicas are supported

In public preview, the Hyperscale databases are created with one read-scale replica (two replicas in total) by default. If you want to add or remove read-scale replicas, please file a support request using the Azure portal.

### For high availability, do I need to provision additional compute nodes

In Hyperscale databases, the high availability is provided at the storage level. You only need one replica to provide high availability. 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 degraded IOPS and query performance.

For mission-critical apps that require high availability, you should provision at least 2 compute nodes including the primary compute node (default). That way there is a hot-standby available in the case of a failover.

## Data size and storage questions

### What is the max db size supported with SQL Database 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, the log generation rate might be throttled for continuous aggressive workloads. The peak and average log generation rate is not yet known (still in preview).

### Does my temp db scale as my database grows

Your `tempdb` database is located on local SSD storage and is configured based on the compute size that you provision. Your `tempdb` is optimized and laid out to provide maximum performance benefits. The `tempdb` size is not configurable and is managed for you by storage sub-system.

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

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

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

5 GB

### In what increments does my database size grow

1 GB

## **Is the storage in SQL Database Hyperscale local or remote**

In Hhyperscale, data files are stored in Azure standard storage. Data is heavily cached on local SSD storage, on machines close to the compute nodes. In addition, compute nodes have a cache on local SSD and in-memory (Buffer Pool, and so on), to reduce the frequency of fetching data from remote nodes.

## **Can I manage or define files or filegroups with Hhyperscale**

No

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

No

## **How are data files laid out with SQL Database Hhyperscale**

The data files are controlled by page servers. As the data size grows, data files and associated page server nodes are added.

## **Is database shrink supported**

No

## **Is database compression supported**

Yes

## **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 a [proportional fill strategy](#) to distribute data over data files.

# Data migration questions

## **Can I move my existing Azure SQL databases to the Hhyperscale service tier**

Yes. You can move your existing Azure SQL databases to Hhyperscale. In public preview, this is a one-way migration. You can't move databases from Hhyperscale to another service tier. We recommend you make a copy of your production databases and migrate to Hhyperscale for proof of concepts (POCs).

## **Can I move my Hhyperscale databases to other editions**

No. In public preview, you can't move a Hhyperscale database to another service tier.

## **Do I lose any functionality or capabilities after migration to the Hhyperscale service tier**

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

## **Can I move my on-premises SQL Server database or my SQL Server virtual machine database to Hhyperscale**

Yes. You can use all existing migration technologies to migrate to Hhyperscale, including BACPAC, transactional replication, logical data loading. See also the [Azure Database Migration Service](#).

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

Downtime is the same as the downtime when you migrate your databases to a single database in Azure SQL Database. You can use [transactional replication](#) to minimize downtime migration for databases up to few TB in size. For very large database (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 SQL Database Hhyperscale**

Not yet known (still in preview)

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

You can read data from Azure Storage and load data into a Hyperscale database (just like you can do with a regular single database). Polybase is currently not supported on Azure SQL Database. You can do Polybase using [Azure Data Factory](#) or running a Spark job in [Azure Databricks](#) with the [Spark connector for SQL](#). The Spark connector to SQL supports bulk insert.

Simple recovery or bulk logging model is not supported in Hyperscale. Full recovery model is required to provide high availability. However, Hyperscale provides a better data ingest rate compared to a single Azure SQL database because of the new log architecture.

#### **Does SQL Database Hyperscale allow provisioning multiple nodes for ingesting large amounts of data**

No. SQL Database Hyperscale is a SMP architecture and is not an asymmetric multiprocessing or a multi-master architecture. You can only create multiple replicas to scale out read-only workloads.

#### **What is the oldest SQL Server version will SQL Database Hyperscale support migration from**

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 SQL Database Hyperscale support migration from other data sources such as Aurora, MySQL, Oracle, DB2, and other database platforms**

Yes. Coming from different data sources other than SQL Server requires logical migration. You can use the [Azure Database Migration Service](#) for a logical migration.

## Business continuity and disaster recovery questions

#### **What SLA's are provided for a Hyperscale database**

In general, an SLA is not provided during public preview. However, Hyperscale provides the same level of high availability with current SQL DB offerings. See [SLA](#).

#### **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 SQL Database Hyperscale databases. Instead, there are regular snapshots of the data files and log that is generated is simply retained as is for the retention period configured or available to you.

#### **Does SQL Database Hyperscale support Point in Time Restore**

Yes

#### **What is the Recovery Point Objective (RPO)/Recovery Time Objective (RTO) with backup/restore in SQL Database Hyperscale**

The RPO is 0 min. The RTO goal is less than 10 minutes, regardless of database size. However, during public preview, you may experience longer restore time.

#### **Do backups of large databases affect compute performance on my primary**

No. Backups are managed by the storage subsystem, and leverage file snapshots. They do not impact the user workload on the primary.

#### **Can I perform geo-restore with a SQL Database Hyperscale database**

No, not during public preview.

#### **Can I setup Geo-Replication with SQL Database Hyperscale database**

No, not during public preview.

#### **Do my secondary compute nodes get geo-replicated with SQL Database Hyperscale**

No, not during public preview.

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

No. The storage format for Hyperscale databases is different from traditional SQL Server, and you don't control backups or have access to them. To take your data out of a SQL Database Hyperscale database, either use the export service or use scripting plus BCP.

## 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 during public preview, including but not limited to TDE and long term retention backup. After you migrate your databases to Hhyperscale, those features stop working.

### **Will Polybase work with SQL Database Hhyperscale**

No. Polybase isn't supported on Azure SQL Database.

### **Does the compute have support for R and python**

No. R and Python are not supported in Azure SQL Database.

### **Are the compute nodes containerized**

No. Your database resides on a compute VM and not a container.

## Performance questions

### **How much throughput can I push on the largest SQL Database Hhyperscale compute**

Not yet known (still in preview)

### **How many IOPS do I get on the largest SQL Database Hhyperscale compute**

Not yet known (still in preview)

### **Does my throughput get affected by backups**

No. Compute is decoupled from the storage layer to avoid impact on compute.

### **Does my throughput get affected as I provision additional compute nodes**

Because the storage is shared and there is no direct physical replication happening between primary and secondary compute nodes, technically, the throughput on primary node will be affected by adding read-scale nodes. However, we may throttle continuous aggressive workload to allow log apply on secondary nodes and page servers to catch up, and avoid bad read performance on secondary nodes.

## Scalability questions

### **How long would it take to scale up and down a compute node**

Several minutes

### **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 failover happens to the compute node with the target size. Adding read replicas does not result in connection drops.

### **Is the scaling up and down of compute nodes automatic or end-user triggered operation**

End-user. Not automatic.

#### **Does my `tempdb` also grow as the compute is scaled up**

Yes. Temp db will scale up automatically as the compute grows.

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

No. Only the primary compute node accepts read/write requests. Secondary compute nodes only accept read-only requests.

## Read scale questions

#### **How many secondary compute nodes can I provision**

In public preview, we create 2 replicas for Hyperscale databases by default. If you want to adjust the number of replicas, please file a support request using the Azure portal.

#### **How do I connect to these secondary compute nodes**

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

#### **Can I create a dedicated endpoint for the read-scale replica**

No. In public preview, you can only connect to read-scale replica by specifying `ApplicationIntent=ReadOnly`.

#### **Does the system do intelligent load balancing of the read workload**

No. In preview, the read only workload is re-directed to a random read-scale replica.

#### **Can I scale up/down the secondary compute nodes independently of the primary compute**

No, not during public preview.

#### **Do I get different temp db sizing for my primary compute and my additional secondary compute nodes**

No. Your `tempdb` is configured based on the compute size provisioning, during public preview, your secondary compute nodes are the same size as the primary compute.

#### **Can I add indexes and views on my secondary compute nodes**

No. Hyperscale databases have shared storage, meaning that all compute nodes see the same tables, indexes and views. If you want additional indexes optimized for reads on secondary – you must add them on the primary first.

#### **How much delay is there going to be between the primary and secondary compute node**

From the time a transaction is committed on the primary, depending on the log generation rate, it can either be instantaneous or in low milliseconds.

## Next steps

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

# DTU-based service tiers

10/23/2018 • 7 minutes to read • [Edit Online](#)

DTU-based service tiers 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 provide flexibility of changing compute sizes without downtime. Single databases and elastic pools are billed hourly based on service tier and compute size.

## IMPORTANT

SQL Database Managed Instance, currently in public preview 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%                     |
| Backup retention            | 7 days                     | 35 days                    | 35 days                    |
| CPU                         | Low                        | Low, Medium, High          | Medium, High               |
| IO throughput (approximate) | 2.5 IOPS per DTU           | 2.5 IOPS per DTU           | 48 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                  |

## NOTE

You can get a free Azure SQL database at the Basic service tier in conjunction with a 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 the following: West Central US, China East, USDoDCentral, Germany Central, USDoDEast, US Gov Southwest, USGov Iowa, Germany Northeast, China North. In other regions, the storage max in the Premium tier is limited to 1 TB. 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**

ASDB 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        |
| 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](#).

# Prepay for SQL Database compute resources with Azure SQL Database reserved capacity

9/26/2018 • 6 minutes to read • [Edit Online](#)

Save money with Azure SQL Database by prepaying for Azure SQL Database compute resources compared to pay-as-you-go prices. With Azure SQL Database reserved capacity, you make an upfront commitment on SQL Database 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 SQL Database 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 pre-paying for the compute costs for the SQL Database instances 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](#). To buy SQL Database reserved capacity:

- You must be in the Owner role for at least one Enterprise or Pay-As-You-Go subscription.
- For Enterprise subscriptions, Azure reservation purchases must be enabled in the [EA portal](#).
- 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 SQL single databases and/or elastic pools 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 suppose 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 SQL Database Single/Elastic Pool General Purpose - Compute Gen5 and a 40 (2x4 + 32) vCore 1 year reservation for SQL Database Single/Elastic Pool Business Critical - Compute 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 Select Product Type pane, select **SQL Database** to purchase a new reservation for SQL Database.
4. Fill in the required fields. Existing or new single databases or elastic pools 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.

[Home](#) > [Reservations](#) > [Create SQL Reserved vCores](#)

## Create SQL Reserved vCores

SQL Reserved vCores provide a significant discount over pay-as-you-go prices by allowing you to pre-pay for the future use of compute capacity for your Azure SQL Database (PaaS) deployments. Additional software costs will still apply. For SQL Server on Azure VMs (IaaS), purchase Reserved Virtual Machines Instances. [Learn more about SQL Reserved Instance](#).

**BASICS**

|              |   |   |
|--------------|---|---|
| * Name       | SQL_Reservation_09-25-2018_10-10  | ✓ |
| Subscription | Pay-As-You-Go   | ▼ |
| Scope        | <input checked="" type="radio"/> Shared <input type="radio"/> Single subscription | ⓘ |

**DETAILS**

|                    |  |     |
|--------------------|--|-----|
| * Region           | Canada Central   | ⓘ   |
| * Deployment Type  | SQL Database Single/Elastic Pool                                     | ⓘ   |
| * Performance Tier | SQL Database Single/Elastic Pool Business Critical - Compute Gen5... | ⓘ   |
| Term               | Three years  | ⓘ   |
| * Quantity         | 24   | ✓ ⓘ |

**COSTS**

|                               |     |  |
|-------------------------------|-----|--|
| Cost per vCore (compute only) | USD |  |
| Quantity                      | 24  | SQL Database Single/Elastic Pool<br>Business Critical - Compute Gen5,<br>vCore |
| * Total Cost                  | USD |  |

**Estimated savings\***

\* Estimate doesn't include SQL License cost, storage and networking.  
 \* Additional taxes may apply.  
 \* Payment will be processed using the payment method on file for the selected subscription.  
 \* Estimated savings are calculated based on your current on-demand rate.

**Purchase**

| FIELD        | DESCRIPTION   |
|--------------|---|
| Name         | The name of this reservation.   |
| Subscription | 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 number: MS-AZR-0017P) or Pay-As-You-Go (offer number: MS-AZR-0003P). For an enterprise subscription, the charges are deducted from the enrollment's monetary commitment balance or charged as overage. For Pay-As-You-Go subscription, the charges are billed to the credit card or invoice payment method on the subscription. |

| FIELD            | DESCRIPTION  |
|------------------|--|
| Scope            | <p>The vCore reservation's scope can cover one subscription or multiple subscriptions (shared scope). If you select:</p> <ul style="list-style-type: none"> <li>• Single subscription - The vCore reservation discount is applied to SQL Database instances in this subscription.</li> <li>• 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 (except dev/test 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.</li> </ul> |
| Region           | The Azure region that's covered by the SQL Database reserved capacity reservation.   |
| 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 number of instances being purchased within the SQL Database reserved capacity reservation. The quantity is the number of running SQL Database instances that can get the billing discount. For example, if you are running 10 SQL Database instances in the East US, then you would specify quantity as 10 to maximize the benefit for all running machines.   |

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

## Cancellations and exchanges

If you need to cancel your SQL Database reserved capacity reservation, there may be a 12% early termination fee. Refunds are based on the lowest price of either your purchase price or the current price of the reservation. Refunds are limited to \$50,000 per year. The refund you receive is the remaining pro-rated balance minus the 12% early termination fee. To request a cancellation, go to the reservation in the Azure portal and select **Refund** to create a support request.

If you need to change your SQL Database reserved capacity reservation to another region, deployment type, performance tier, or term, you can exchange it for another reservation that's of equal or greater value. The term start date for the new reservation doesn't carry over from the exchanged reservation. The 1 or 3 year term starts from when you create the new reservation. To request an exchange, go to the reservation in the Azure portal, and select **Exchange** to create a support request.

## 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.

## 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](#)

## Need help? Contact support

If you still have further questions, [contact support](#) to get your issue resolved quickly.

# Azure SQL Database logical servers and their management

10/19/2018 • 10 minutes to read • [Edit Online](#)

## What is an Azure SQL logical server

A logical server acts as a central administrative point for multiple single or [pooled](#) databases, [logins](#), [firewall rules](#), [auditing rules](#), [threat detection policies](#), and [failover groups](#). A logical server can be in a different region than its resource group. The logical server must exist before you can create the Azure SQL database. All databases on a server are created within the same region as the logical server.

A logical server is a logical construct that 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 their logical servers, and exposes no instance-level access or features. In contrast, a server in a SQL Database Managed Instance is similar to a SQL Server instance that you may be familiar with in the on-premises world.

When you create a logical 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.

An Azure Database logical 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 user databases created on a logical 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 server (logical 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 logical 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.

#### 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 'Firewall settings' section of the Azure portal for a specific database. It includes fields for 'Allow access to Azure services' (ON), 'Client IP address' (73.42.249.7), and a table for defining IP rules. A 'DTU' performance chart is visible on the left.

#### 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

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  |
|---|--|
| <a href="#">New-AzureRmSqlDatabase</a>    | Creates a database   |
| <a href="#">Get-AzureRmSqlDatabase</a>    | Gets one or more databases   |
| <a href="#">Set-AzureRmSqlDatabase</a>    | Sets properties for a database, or moves an existing database into an elastic pool |
| <a href="#">Remove-AzureRmSqlDatabase</a> | Removes a database   |
| <a href="#">New-AzureRmResourceGroup</a>  | Creates a resource group   |

| CMDLET                                 | DESCRIPTION   |
|--|---|
| New-AzureRmSqlServer                   | Creates a server  |
| Get-AzureRmSqlServer                   | Returns information about servers   |
| Set-AzureRmSqlServer                   | Modifies properties of a server   |
| Remove-AzureRmSqlServer                | Removes a server  |
| New-AzureRmSqlServerFirewallRule       | Creates a server-level firewall rule  |
| Get-AzureRmSqlServerFirewallRule       | Gets firewall rules for a server  |
| Set-AzureRmSqlServerFirewallRule       | Modifies a firewall rule in a server  |
| Remove-AzureRmSqlServerFirewallRule    | Deletes a firewall rule from a server.  |
| New-AzureRmSqlServerVirtualNetworkRule | 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 a single Azure SQL database using PowerShell](#). For PowerShell example scripts, see [Use PowerShell to create a single Azure SQL database and configure a firewall rule](#) and [Monitor and scale a single SQL 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 |
| az sql db list-editions | Lists available service objectives and storage limits                                    |
| az sql db list-usages   | Returns database usages  |
| az sql db show          | Gets a database or data warehouse  |
| az sql db update        | Updates a database   |
| az sql db delete        | Removes a database   |

| CMDLET   | DESCRIPTION                          |
|--|--------------------------------------|
| <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 a single Azure SQL database using the Azure CLI](#). For Azure CLI example scripts, see [Use CLI to create a single Azure SQL database and configure a firewall rule](#) and [Use CLI to monitor and scale a single SQL 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.  |

| COMMAND   | DESCRIPTION  |
|---|--|
| <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.  |
| <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.                     |
| <a href="#">Firewall Rules - Delete</a>           | Deletes a firewall rule.                                |
| <a href="#">Firewall Rules - Get</a>              | Gets a firewall rule.                                   |
| <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](#).

# SQL Database resource limits for single and pooled databases on a logical server

10/5/2018 • 3 minutes to read • [Edit Online](#)

This article provides an overview of the SQL Database resource limits for single and pooled databases on a logical server. It also provides information regarding what happens when those resource limits are hit or exceeded.

## 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 |

## NOTE

To obtain more DTU /eDTU quota, vCore quota, or more servers than the default amount, a new support request can be submitted in the Azure portal for the subscription with issue type "Quota". The DTU / eDTU quota and database limit per server constrains the number of elastic pools per server.

## IMPORTANT

As the number of databases approaches the limit per logical 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.

## 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 can even time out. Under these conditions, queries may be queued by the service and are provided resources for execution as resource 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](#). Database SELECTS and DELETES continue to succeed.

When encountering high space utilization, mitigation options include:

- Increasing the max size of the database or elastic pool, or add 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 and eDTUs). 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.

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](#).

## Next steps

- See [SQL Database FAQ](#) for answers to frequently asked questions.
- 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](#).

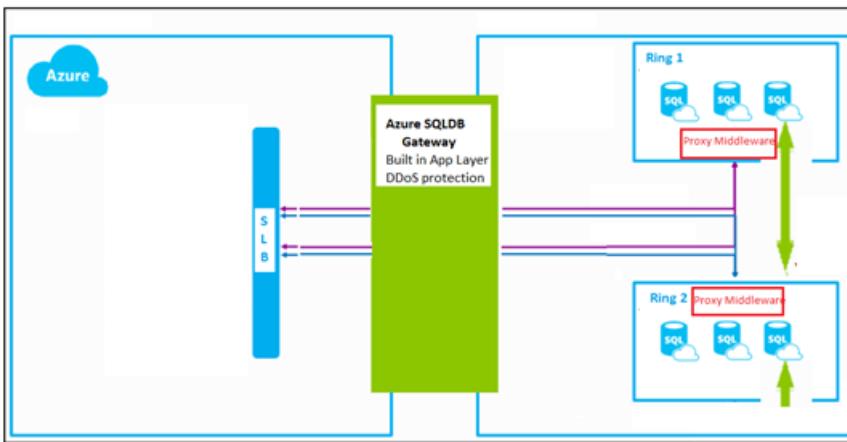
# Azure SQL Database Connectivity Architecture

10/24/2018 • 6 minutes to read • [Edit Online](#)

This article explains the Azure SQL Database connectivity architecture and explains how the different components function to direct traffic to your instance of Azure SQL Database. These Azure SQL Database connectivity components function to direct network traffic to the Azure database 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 through the Azure SQL Database software load-balancer (SLB) and the Azure SQL Database gateway.

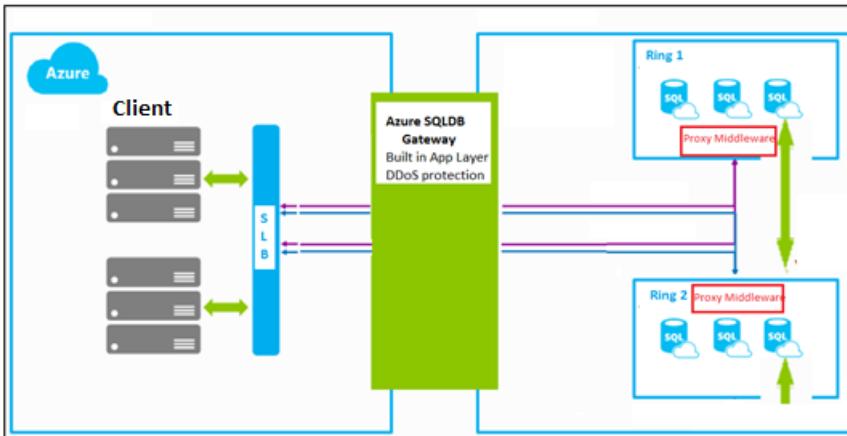
- Clients within Azure or outside of Azure connect to the SLB, which has a public IP address and listens on port 1433.
- The SLB directs traffic to the Azure SQL Database gateway.
- The gateway redirects the traffic to the correct proxy middleware.
- The proxy middleware redirects the traffic to the appropriate Azure SQL database.

### IMPORTANT

Each of these components has distributed denial of service (DDoS) protection built-in at the network and the app layer.

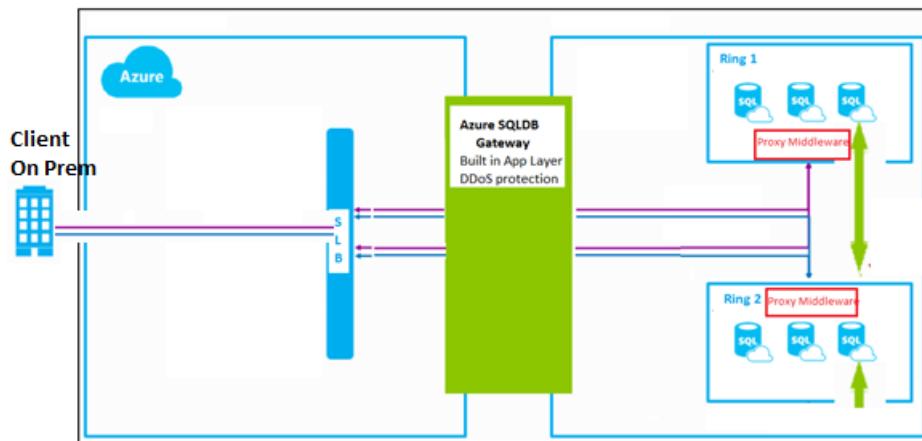
## Connectivity from within Azure

If you are connecting from within Azure, your connections have a connection policy of **Redirect** by default. A policy of **Redirect** means that connections after the TCP session is established to the Azure SQL database, the client session is then redirected to the proxy middleware with a change to the destination virtual IP from that of the Azure SQL Database gateway to that of the proxy middleware. Thereafter, all subsequent packets flow directly via the proxy middleware, 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

When using service endpoints with Azure SQL Database your policy is **Proxy** by default. To enable connectivity from inside your VNet, you must allow outbound connections to the Azure SQL Database Gateway IP addresses specified in the list below.

When using service endpoints we highly recommend changing your connection policy to **Redirect** to enable better performance. If you change your connection policy to **Redirect** it will not be sufficient to allow outbound on your NSG to Azure SQL Database gateway IPs listed below, you must allow outbound to all Azure SQL Database IPs. This can be accomplished with the help of NSG (Network Security Groups) Service Tags. For more information, see [Service Tags](#).

## Azure SQL Database gateway IP addresses

To connect to an Azure SQL database from on-premises resources, you need to allow outbound network traffic to the Azure SQL Database gateway for your Azure region. Your connections only go via the gateway when connecting in Proxy mode, which is the default when connecting from on-premises resources.

The following table lists the primary and secondary IPs of the Azure SQL Database gateway for all data regions. For some regions, there are two IP addresses. In these regions, the primary IP address is the current IP address of the gateway and the second IP address is a failover IP address. The failover address is the address to which we might move your server to keep the service availability high. For these regions, we recommend that you allow

outbound to both the IP addresses. The second IP address is owned by Microsoft and does not listen in on any services until it is activated by Azure SQL Database to accept connections.

#### IMPORTANT

If you are connecting from within Azure your connection policy will be **Redirect** by default (except if you are using service endpoints). It will not be sufficient to allow the following IPs. You must allow all Azure SQL Database IPs. If you are connecting from within a VNet, this can be accomplished with the help of NSG (Network Security Groups) Service Tags. For more information, see [Service Tags](#).

| REGION NAME          | PRIMARY IP ADDRESS | SECONDARY IP ADDRESS |
|----------------------|--------------------|----------------------|
| Australia East       | 191.238.66.109     | 13.75.149.87         |
| Australia South East | 191.239.192.109    | 13.73.109.251        |
| Brazil South         | 104.41.11.5        |                      |
| Canada Central       | 40.85.224.249      |                      |
| Canada East          | 40.86.226.166      |                      |
| Central US           | 23.99.160.139      | 13.67.215.62         |
| China East 1         | 139.219.130.35     |                      |
| China East 2         | 40.73.82.1         |                      |
| China North 1        | 139.219.15.17      |                      |
| China North 2        | 40.73.50.0         |                      |
| East Asia            | 191.234.2.139      | 52.175.33.150        |
| East US 1            | 191.238.6.43       | 40.121.158.30        |
| East US 2            | 191.239.224.107    | 40.79.84.180 *       |
| 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           | 191.237.240.43     | 13.78.61.196         |

| Region Name      | Primary IP Address | Secondary IP Address |
|------------------|--------------------|----------------------|
| Japan West       | 191.238.68.11      | 104.214.148.156      |
| Korea Central    | 52.231.32.42       |                      |
| Korea South      | 52.231.200.86      |                      |
| North Central US | 23.98.55.75        | 23.96.178.199        |
| North Europe     | 191.235.193.75     | 40.113.93.91         |
| South Central US | 23.98.162.75       | 13.66.62.124         |
| South East Asia  | 23.100.117.95      | 104.43.15.0          |
| UK North         | 13.87.97.210       |                      |
| UK South 1       | 51.140.184.11      |                      |
| UK South 2       | 13.87.34.7         |                      |
| UK West          | 51.141.8.11        |                      |
| West Central US  | 13.78.145.25       |                      |
| West Europe      | 191.237.232.75     | 40.68.37.158         |
| West US 1        | 23.99.34.75        | 104.42.238.205       |
| West US 2        | 13.66.226.202      |                      |

\* **NOTE:** East US 2 has also a tertiary IP address of [52.167.104.0](#).

## 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 middleware proxy. For this setting, you need to allow outbound to multiple IPs.

## Script to change connection settings via PowerShell

### IMPORTANT

This script requires the [Azure PowerShell module](#).

The following PowerShell script shows how to change the connection policy.

```
Connect-AzureRmAccount
Select-AzureRmSubscription -SubscriptionName <Subscription Name>

# Azure Active Directory ID
$tenantId = "<Azure Active Directory GUID>"
$authUrl = "https://login.microsoftonline.com/$tenantId"

# Subscription ID
$subscriptionId = "<Subscription GUID>"

# Create an App Registration in Azure Active Directory. Ensure the application type is set to NATIVE
# Under Required Permissions, add the API: Windows Azure Service Management API

# Specify the redirect URL for the app registration
$uri = "<NATIVE APP - REDIRECT URI>"

# Specify the application id for the app registration
$clientId = "<NATIVE APP - APPLICATION ID>"

# Logical SQL Server Name
$serverName = "<LOGICAL DATABASE SERVER - NAME>"

# Resource Group where the SQL Server is located
$resourceGroupName= "<LOGICAL DATABASE SERVER - RESOURCE GROUP NAME>"

# Login and acquire a bearer token
$AuthContext = [Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext]$authUrl
$result = $AuthContext.AcquireToken(
"https://management.core.windows.net/",
ClientId,
[Uri]$uri,
[Microsoft.IdentityModel.Clients.ActiveDirectory.PromptBehavior]::Auto
)

$authHeader = @{
'Content-Type'='application\json;
'Authorization'=$result.CreateAuthorizationHeader()
}

#Get current connection Policy
Invoke-RestMethod -Uri
"https://management.azure.com/subscriptions/$subscriptionId/resourceGroups/$resourceGroupName/providers/Microsoft.Sql/servers/$serverName/connectionPolicies/Default?api-version=2014-04-01-preview" -Method GET -Headers
$authHeader

#Set connection policy to Proxy
$connectionType="Proxy" <#Redirect / Default are other options#>
$body = @{properties=@{connectionType=$connectionType}} | ConvertTo-Json

# Apply Changes
Invoke-RestMethod -Uri
"https://management.azure.com/subscriptions/$subscriptionId/resourceGroups/$resourceGroupName/providers/Microsoft.Sql/servers/$serverName/connectionPolicies/Default?api-version=2014-04-01-preview" -Method PUT -Headers
$authHeader -Body $body -ContentType "application/json"
```

## Script to change connection settings via Azure CLI

### IMPORTANT

This script requires the [Azure CLI](#).

The following CLI script shows how to change the connection policy.

```
<pre>
# Get SQL Server ID
sqlserverid=$(az sql server show -n <b>sql-server-name</b> -g <b>sql-server-group</b> --query 'id' -o tsv)

# Set URI
id="$sqlserverid/connectionPolicies/Default"

# Get current connection policy
az resource show --ids $id

# Update connection policy
az resource update --ids $id --set properties.connectionType=Proxy

</pre>
```

## 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](#).

# Resolving Transact-SQL differences during migration to SQL Database

10/15/2018 • 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 a 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 a 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.

# Replication to SQL Database single and pooled databases

10/8/2018 • 3 minutes to read • [Edit Online](#)

SQL Server replication can be configured to single and pooled databases on a [logical 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

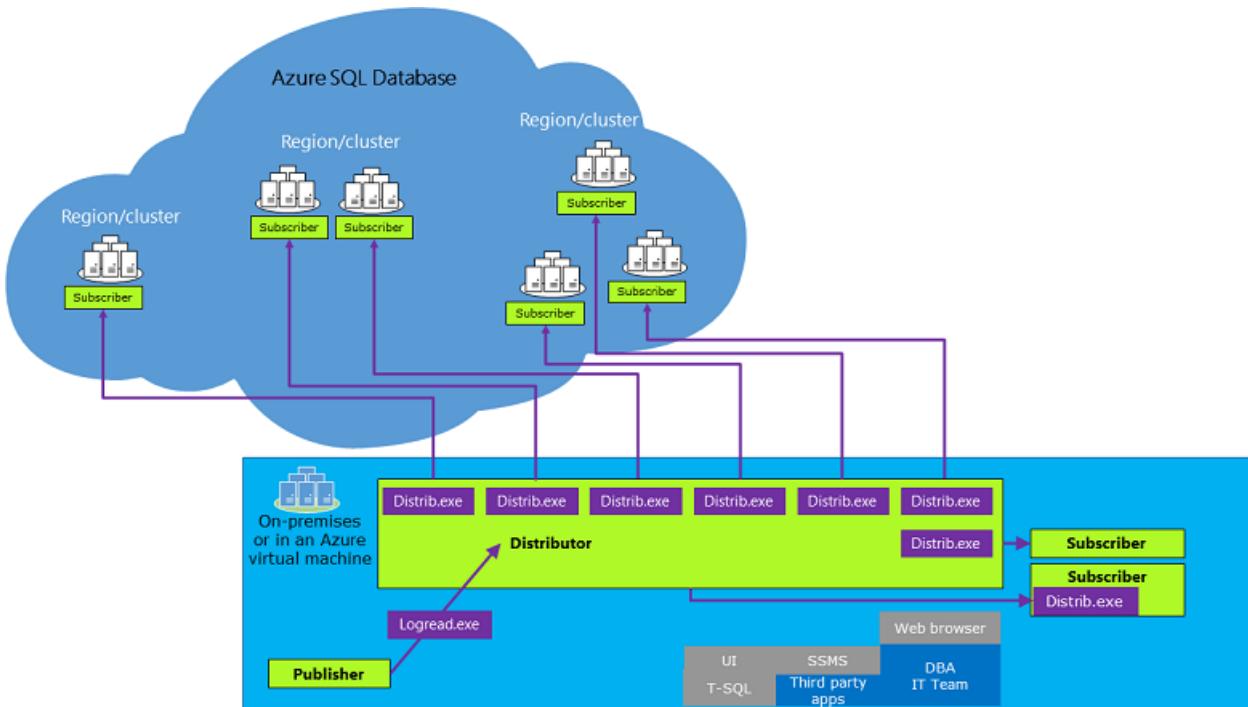
- The publisher and distributor must be at least at one of the following versions:
  - SQL Server 2017 (14.x)
  - SQL Server 2016 (13.x)
  - SQL Server 2014 (12.x) SP1 CU3
  - SQL Server 2014 (12.x) RTM CU10
  - SQL Server 2012 (11.x) SP2 CU8 or SP3
- Attempting to configure replication using an older 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. The initial data set is typically a snapshot that is created by the Snapshot Agent and distributed and applied by the Distribution Agent. The initial data set can also be supplied through a backup or other means, such as SQL Server Integration Services.

### 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 logical server name as the subscriber (for example **N'azuresqldbns.database.windows.net'**) and the Azure SQL database name as the destination database (for example **AdventureWorks**).

## See Also

- [Create a Publication](#)
- [Create a Push Subscription](#)
- [Types of Replication](#)
- [Monitoring \(Replication\)](#)
- [Initialize a Subscription](#)

# Create and manage logical servers and single databases in Azure SQL Database

10/19/2018 • 7 minutes to read • [Edit Online](#)

You can create and manage Azure SQL database logical servers and single databases using the Azure portal, PowerShell, Azure CLI, REST API, and Transact-SQL.

## Azure portal: Manage logical servers and 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 server (logical 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 logical 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 Microsoft Azure portal interface for creating a new SQL Database. On the left, the sidebar includes links for Favorites, Dashboard, All resources, SQL servers, SQL databases, Resource groups, App Services, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Activity log, Security Center, Cost Management + B..., Help + support, and Azure Active Directory. A red box highlights the '+ Create a resource' button. The main content area shows the 'New' blade for 'SQL Database'. It includes a search bar, a 'Search the Marketplace' input, and a list of popular services like Windows Server 2016 VM, Ubuntu Server 16.04 LTS VM, Web App, Containers, Databases, Data + Analytics, AI + Cognitive Services, Internet of Things, Enterprise Integration, Security + Identity, Developer tools, Monitoring + Management, Add-ons, and Blockchain. A red box highlights the 'Databases' item. To the right, the 'SQL Database' configuration form is shown with fields for 'Database name' (set to 'mySampleDatabase'), 'Subscription' (dropdown), 'Resource group' (dropdown, with 'Create new' selected), 'Select source' (dropdown set to 'Sample (AdventureWorksLT)'), 'Server' (dropdown), and 'Configure required settings' (button). Below the form, there are sections for 'Want to use SQL elastic pool?' (radio buttons for 'Yes' and 'Not now'), 'Pricing tier' (dropdown with a warning icon), 'Collation' (dropdown set to 'SQL\_Latin1\_General\_CI\_AS'), and options to 'Pin to dashboard' and 'Create' (button). The bottom right corner of the screenshot has a small '...' icon.

## IMPORTANT

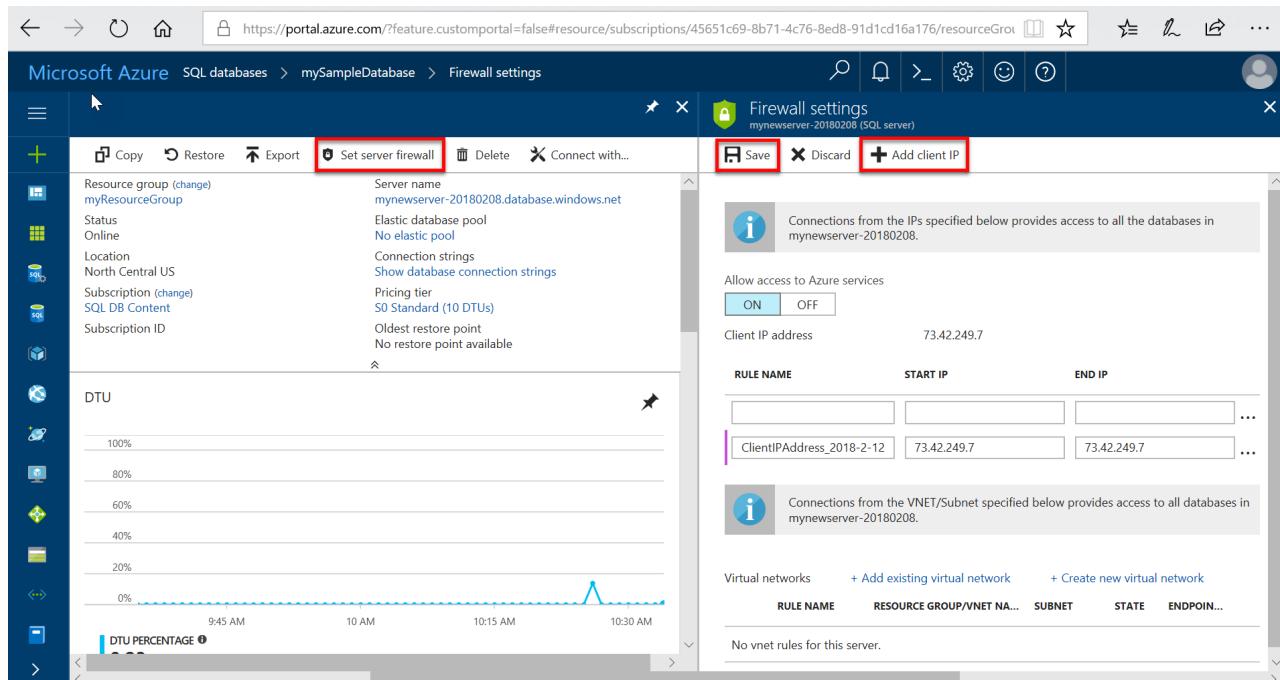
For information on selecting the pricing tier for your database, see [DTU-based purchasing model](#) and [vCore-based purchasing model](#).

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 Microsoft Azure portal interface for managing a SQL database. On the left, there's a sidebar with various icons and a DTU usage chart. The main area shows the 'Firewall settings' for a specific server. At the top, there are buttons for 'Copy', 'Restore', 'Export', 'Set server firewall' (which is highlighted with a red box), 'Delete', and 'Connect with...'. Below these are server details like name, elastic pool, connection strings, and pricing tier. The right side shows the 'Firewall settings' configuration, including a table for client IP rules and a section for virtual networks. The 'Save' and 'Add client IP' buttons are also highlighted with red boxes.

## 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 logical servers and databases

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](#).

**TIP**

For a PowerShell quickstart, see [Create a single Azure SQL database using PowerShell](#). For PowerShell example scripts, see [Use PowerShell to create a single Azure SQL database and configure a firewall rule](#) and [Monitor and scale a single SQL database using PowerShell](#).

| CMDLET   | DESCRIPTION   |
|--|---|
| <a href="#">New-AzureRmSqlDatabase</a>                 | Creates a database  |
| <a href="#">Get-AzureRmSqlDatabase</a>                 | Gets one or more databases  |
| <a href="#">Set-AzureRmSqlDatabase</a>                 | Sets properties for a database, or moves an existing database into an elastic pool                    |
| <a href="#">Remove-AzureRmSqlDatabase</a>              | Removes a database  |
| <a href="#">New-AzureRmResourceGroup</a>               | Creates a resource group  |
| <a href="#">New-AzureRmSqlServer</a>                   | Creates a server  |
| <a href="#">Get-AzureRmSqlServer</a>                   | Returns information about servers   |
| <a href="#">Set-AzureRmSqlServer</a>                   | Modifies properties of a server   |
| <a href="#">Remove-AzureRmSqlServer</a>                | Removes a server  |
| <a href="#">New-AzureRmSqlServerFirewallRule</a>       | Creates a server-level firewall rule  |
| <a href="#">Get-AzureRmSqlServerFirewallRule</a>       | Gets firewall rules for a server  |
| <a href="#">Set-AzureRmSqlServerFirewallRule</a>       | Modifies a firewall rule in a server  |
| <a href="#">Remove-AzureRmSqlServerFirewallRule</a>    | Deletes a firewall rule from a server.  |
| <a href="#">New-AzureRmSqlServerVirtualNetworkRule</a> | Creates a <i>virtual network rule</i> , based on a subnet that is a Virtual Network service endpoint. |

## Azure CLI: Manage logical servers and 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 a single Azure SQL database using the Azure CLI](#). For Azure CLI example scripts, see [Use CLI to create a single Azure SQL database and configure a firewall rule](#) and [Use CLI to monitor and scale a single SQL 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 logical servers and 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 (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.  |

| 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 logical servers and 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.                              |
| <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                       |
|---------------------------------|-----------------------------------|
| 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](#).

# Azure SQL Database vCore-based purchasing model limits for a single database

10/16/2018 • 7 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database single databases using the vCore-based purchasing model.

For DTU-based purchasing model limits for single databases on a logical server, see [Overview of resource limits on a logical server](#).

## 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 for a single database using the [Azure portal](#), [Transact-SQL](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

## General Purpose service tier: Storage sizes and compute sizes

### Generation 4 compute platform

| COMPUTE SIZE                | GP_GEN4_1                        | GP_GEN4_2                        | GP_GEN4_4                        | GP_GEN4_8                        | GP_GEN4_16                       | GP_GEN4_24                       |
|-----------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| H/W generation              | 4                                | 4                                | 4                                | 4                                | 4                                | 4                                |
| vCores                      | 1                                | 2                                | 4                                | 8                                | 16                               | 24                               |
| Memory (GB)                 | 7                                | 14                               | 28                               | 56                               | 112                              | 168                              |
| 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                              |
| Storage type                | Premium (Remote) Storage         |
| IO latency (approximate)    | 5-7 ms (write)<br>5-10 ms (read) |
| Max data size (GB)          | 1024                             | 1024                             | 1536                             | 3072                             | 4096                             | 4096                             |

## Generation 5 compute platform

| <b>COMPUTE SIZE</b>               | <b>GP_GEN5_2</b>                 | <b>GP_GEN5_4</b>                 | <b>GP_GEN5_8</b>                 | <b>GP_GEN5_16</b>                | <b>GP_GEN5_24</b>                | <b>GP_GEN5_32</b>                | <b>GP_GEN5_40</b>                | <b>GP_GEN5_80</b>                |
|-----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| Storage type                      | Premium (Remote) Storage         |
| IO latency (approximate)          | 5-7 ms (write)<br>5-10 ms (read) |
| Max data size (GB)                | 1024                             | 1024                             | 1536                             | 3072                             | 4096                             | 4096                             | 4096                             | 4096                             |
| Max log size (GB)                 | 307                              | 307                              | 461                              | 614                              | 1229                             | 1229                             | 1229                             | 1229                             |
| TempDB size (GB)                  | 64                               | 128                              | 256                              | 384                              | 384                              | 384                              | 384                              | 384                              |
| Target IOPS (64 KB)               | 500                              | 1000                             | 2000                             | 4000                             | 6000                             | 7000                             | 7000                             | 7000                             |
| Max concurrent workers (requests) | 200                              | 400                              | 800                              | 1600                             | 2400                             | 3200                             | 4000                             | 8000                             |
| Max allowed sessions              | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            |
| Number of replicas                | 1                                | 1                                | 1                                | 1                                | 1                                | 1                                | 1                                | 1                                |
| Multi-AZ                          | N/A                              |
| Read Scale-out                    | N/A                              |
| Included backup storage           | 1X DB size                       |
|                                   |                                  |                                  |                                  |                                  |                                  |                                  |                                  |                                  |

## Business Critical service tier: Storage sizes and compute sizes

### Generation 4 compute platform

| <b>COMPUTE SIZE</b> | <b>BC_GEN4_1</b> | <b>BC_GEN4_2</b> | <b>BC_GEN4_4</b> | <b>BC_GEN4_8</b> | <b>BC_GEN4_16</b> | <b>BC_GEN4_24</b> |
|---------------------|------------------|------------------|------------------|------------------|-------------------|-------------------|
| H/W generation      | 4                | 4                | 4                | 4                | 4                 | 4                 |

| <b>COMPUTE SIZE</b>               | <b>BC_GEN4_1</b>                | <b>BC_GEN4_2</b>                | <b>BC_GEN4_4</b>                | <b>BC_GEN4_8</b>                | <b>BC_GEN4_16</b>               | <b>BC_GEN4_24</b>               |
|-----------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|
| vCores                            | 1                               | 2                               | 4                               | 8                               | 16                              | 24                              |
| Memory (GB)                       | 7                               | 14                              | 28                              | 56                              | 112                             | 168                             |
| Columnstore support               | N/A                             | N/A                             | N/A                             | N/A                             | N/A                             | N/A                             |
| In-memory OLTP storage (GB)       | 1                               | 2                               | 4                               | 8                               | 20                              | 36                              |
| 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 size (GB)                  | 32                              | 64                              | 128                             | 256                             | 384                             | 384                             |
| Target IOPS (64 KB)               | 5000                            | 10000                           | 20000                           | 40000                           | 80000                           | 120000                          |
| IO latency (approximate)          | 1-2 ms (write)<br>1-2 ms (read) |
| Max concurrent workers (requests) | 200                             | 400                             | 800                             | 1600                            | 3200                            | 4800                            |
| Max allowed sessions              | 30000                           | 30000                           | 30000                           | 30000                           | 30000                           | 30000                           |
| Number of replicas                | 3                               | 3                               | 3                               | 3                               | 3                               | 3                               |
| Multi-AZ                          | N/A                             | N/A                             | N/A                             | N/A                             | N/A                             | N/A                             |
| Read Scale-out                    | Yes                             | Yes                             | Yes                             | Yes                             | Yes                             | Yes                             |
| Included backup storage           | 1X DB size                      |

### Generation 5 compute platform



| <b>COMPUTE SIZE</b>     | <b>BC_GEN5_2</b> | <b>BC_GEN5_4</b> | <b>BC_GEN5_8</b> | <b>BC_GEN5_16</b> | <b>BC_GEN5_24</b> | <b>BC_GEN5_32</b> | <b>BC_GEN5_40</b> | <b>BC_GEN5_80</b> |
|-------------------------|------------------|------------------|------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| Read Scale-out          | N/A              | 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        | 1X DB size        |
|                         |                  |                  |                  |                   |                   |                   |                   |                   |

## Hyperscale service tier (preview)

### Generation 4 compute platform: Storage sizes and compute sizes

| <b>PERFORMANCE LEVEL</b>          | <b>HS_GEN4_1</b> | <b>HS_GEN4_2</b> | <b>HS_GEN4_4</b> | <b>HS_GEN4_8</b> | <b>HS_GEN4_16</b> | <b>HS_GEN4_24</b> |
|-----------------------------------|------------------|------------------|------------------|------------------|-------------------|-------------------|
| H/W generation                    | 4                | 4                | 4                | 4                | 4                 | 4                 |
| vCores                            | 1                | 2                | 4                | 8                | 16                | 24                |
| Memory (GB)                       | 7                | 14               | 28               | 56               | 112               | 168               |
| 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               |
| Storage type                      | Local SSD         | Local SSD         |
| Max data size (TB)                | 100              | 100              | 100              | 100              | 100               | 100               |
| Max log size (TB)                 | 1                | 1                | 1                | 1                | 1                 | 1                 |
| TempDB size (GB)                  | 32               | 64               | 128              | 256              | 384               | 384               |
| Target IOPS (64 KB)               | To be determined  | To be determined  |
| IO latency (approximate)          | To be determined  | To be determined  |
| Max concurrent workers (requests) | 200              | 400              | 800              | 1600             | 3200              | 4800              |

| PERFORMANCE LEVEL       | HS_GEN4_1 | HS_GEN4_2 | HS_GEN4_4 | HS_GEN4_8 | HS_GEN4_16 | HS_GEN4_24 |
|-------------------------|-----------|-----------|-----------|-----------|------------|------------|
| Max allowed sessions    | 30000     | 30000     | 30000     | 30000     | 30000      | 30000      |
| Number of replicas      | 2         | 2         | 2         | 2         | 2          | 2          |
| Multi-AZ                | N/A       | N/A       | N/A       | N/A       | N/A        | N/A        |
| Read Scale-out          | Yes       | Yes       | Yes       | Yes       | Yes        | Yes        |
| Included backup storage | 7         | 7         | 7         | 7         | 7          | 7          |
|                         |           |           |           |           |            |            |

### Generation 5 compute platform

| PERFORMANCE LEVEL           | HS_GEN5_2 | HS_GENS_4 | HS_GEN5_8 | HS_GEN5_16 | HS_GENS_24 | HS_GEN5_32 | HS_GENS_40 | HS_GEN5_80 |
|-----------------------------|-----------|-----------|-----------|------------|------------|------------|------------|------------|
| H/W generation              | 5         | 5         | 5         | 5          | 5          | 5          | 5          | 5          |
| vCores                      | 2         | 4         | 8         | 16         | 24         | 32         | 40         | 80         |
| Memory (GB)                 | 11        | 22        | 44        | 88         | 132        | 176        | 220        | 440        |
| Columnstore support         | Yes       | 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        | N/A        |
| Storage type                | Local SSD | Local SSD | Local SSD | Local SSD  | Local SSD  | Local SSD  | Local SSD  | Local SSD  |
| Max data size (TB)          | 100       | 100       | 100       | 100        | 100        | 100        | 100        | 100        |
| Max log size (TB)           | 1         | 1         | 1         | 1          | 1          | 1          | 1          | 1          |
| TempDB size (GB)            | 64        | 128       | 256       | 384        | 384        | 384        | 384        | 384        |

| PERFORMANCE LEVEL                       | HS_GEN5_2        | HS_GEN5_4        | HS_GEN5_8        | HS_GEN5_16       | HS_GEN5_24       | HS_GEN5_32       | HS_GEN5_40       | HS_GEN5_80       |
|---|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| Target IOPS (64 KB)                     | To be determined |
| IO latency (approximate)                | To be determined |
| Max concurrent workers (requests)       | 200              | 400              | 800              | 1600             | 2400             | 3200             | 4000             | 8000             |
| Max allowed sessions                    | 30000            | 30000            | 30000            | 30000            | 30000            | 30000            | 30000            | 30000            |
| Number of replicas                      | 2                | 2                | 2                | 2                | 2                | 2                | 2                | 2                |
| Multi-AZ                                | N/A              |
| Read Scale-out                          | Yes              |
| Included backup storage (preview limit) | 7                | 7                | 7                | 7                | 7                | 7                | 7                | 7                |
|   |                  |                  |                  |                  |                  |                  |                  |                  |

## Next steps

- See [SQL Database FAQ](#) for answers to frequently asked questions.
- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).

# Resource limits for single databases using the DTU-based purchasing model

10/29/2018 • 9 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database single databases using the DTU-based purchasing model.

For DTU-based purchasing model resource limits for elastic pools, see [DTU-based resource limits - elastic pools](#). For vCore-based resource limits, see [vCore-based resource limits - single databases](#) and [vCore-based resource limits - elastic pools](#). For more information regarding the different purchasing models, see [Purchasing models and service tiers](#).

## 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](#).

## Single database: Storage sizes and compute sizes

For single databases, 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](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

### 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   |

### Standard service tier

| COMPUTE SIZE          | S0  | S1  | S2  | S3  |
|-----------------------|-----|-----|-----|-----|
| Max DTUs              | 10  | 20  | 50  | 100 |
| Included storage (GB) | 250 | 250 | 250 | 250 |

| <b>COMPUTE SIZE</b>               | <b>S0</b> | <b>S1</b> | <b>S2</b> | <b>S3</b>           |
|-----------------------------------|-----------|-----------|-----------|---------------------|
| 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                |

#### 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         |

| Compute size                      | P1    | P2    | P4    | P6    | P11   | P15   |
|-----------------------------------|-------|-------|-------|-------|-------|-------|
| Max concurrent workers (requests) | 200   | 400   | 800   | 1600  | 2400  | 6400  |
| Max concurrent sessions           | 30000 | 30000 | 30000 | 30000 | 30000 | 30000 |

#### IMPORTANT

More than 1 TB of storage in the Premium tier is currently available in all regions except the following: China East, China North, Germany Central, Germany Northeast, UK North, UK South, US DOD Central, US DOD East, US Government Central, and West Central US. In these regions, the storage max in the Premium tier is limited to 1 TB. See [P11-P15 Current Limitations](#).

## Single database: Change storage size

- 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](#).

## Single database: Change DTUs

After initially picking a service tier, compute size, and storage amount, 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.

Changing the service tier and/or compute size of a database creates a replica of the original database at the new compute size, and then switches connections over to the replica. No data is lost during this process but during the brief moment when we switch over to the replica, connections to the database are disabled, so some transactions in flight may be rolled back. The length of time for the switch-over varies, but is less than 30 seconds 99% of the time. If there are large numbers of transactions in flight at the moment connections are disabled, the length of time for the switch-over may be longer.

The duration of the entire scale-up process depends on both the size and service tier of the database before and after the change. For example, a 250-GB database that is changing to, from, or within a Standard service tier, should complete within six hours. For a database the same size that is changing compute sizes within the Premium service tier, the scale-up should complete within three hours.

**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-AzureRmSqlDatabaseActivity` and `Stop-AzureRmSqlDatabaseActivity`.

- 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 500 GB and its included storage amount is only 250 GB. So, the extra storage amount is  $500\text{ GB} - 250\text{ GB} = 250\text{ GB}$ . 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.
- 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, upgrading the secondary database first is required.
- 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, downgrading the primary database first is required.
- 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.

## Single database: Limitations of P11 and P15 when the maximum size greater than 1 TB

The following considerations and limitations apply to P11 and P15 databases with a maximum size greater than 1 TB:

- If you choose a maximum size greater than 1 TB when creating a database (using a value of 4 TB or 4096 GB), the create command fails with an error if the database is provisioned in an unsupported region.
- For existing P11 and P15 databases located in one of the supported regions, you can increase the maximum storage to beyond 1 TB in increments of 256 GB up to 4 TB. To see if a larger size is supported in your region, use the [DATABASEPROPERTYEX](#) function or inspect the database size in the Azure portal. Upgrading an existing P11 or P15 database can only be performed by a server-level principal login or by members of the dbmanager database role.
- If an upgrade operation is executed in a supported region the configuration is updated immediately. The database remains online during the upgrade process. However, you cannot utilize the full amount of storage beyond 1 TB of storage until the actual database files have been upgraded to the new maximum size. The length of time required depends upon on the size of the database being upgraded.
- When creating or updating a P11 or P15 database, you can only choose between 1-TB and 4-TB maximum size in increments of 256 GB. When creating a P11/P15, the default storage option of 1 TB is pre-selected. For databases located in one of the supported regions, you can increase the storage maximum to up to a maximum of 4 TB for a new or existing single database. For all other regions, the maximum size cannot be increased above 1 TB. The price does not change when you select 4 TB of included storage.
- If the maximum size of a database is set to greater than 1 TB, then it cannot be changed to 1 TB even if the actual storage used is below 1 TB. Thus, you cannot downgrade a P11 or P15 with a maximum size larger than

1 TB to a 1 TB P11 or 1 TB P15 or lower compute size, such as P1-P6). This restriction also applies to the restore and copy scenarios including point-in-time, geo-restore, long-term-backup-retention, and database copy. Once a database is configured with a maximum size greater than 1 TB, all restore operations of this database must be run into a P11/P15 with a maximum size greater than 1 TB.

- 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 sizes 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

- See [SQL Database FAQ](#) for answers to frequently asked questions.
- See [Overview of resource limits on a logical server](#) for information about limits at the server and subscription levels.
- 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 [SQL Database tempdb limits](#).

# Scale single database resources in Azure SQL Database

10/19/2018 • 9 minutes to read • [Edit Online](#)

This article describes how to scale the compute and storage resources available for a single database in Azure SQL Database.

## vCore-based purchasing model: Change storage size

- 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](#).

## vCore-based purchasing model: Change compute resources

After initially picking the number of vCores, 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](#).

Changing the service tier and/or compute size of a database creates a replica of the original database at the new compute size, and then switches connections over to the replica. No data is lost during this process but during the brief moment when we switch over to the replica, connections to the database are disabled, so some transactions in flight may be rolled back. The length of time for the switch-over varies, but is generally under 4 seconds is less than 30 seconds 99% of the time. If there are large numbers of transactions in flight at the moment connections are disabled, the length of time for the switch-over may be longer.

The duration of the entire scale-up process depends on both the size and service tier of the database before and after the change. For example, a 250-GB database that is changing to, from, or within a General Purpose service tier, should complete within six hours. For a database the same size that is changing compute sizes within the Business Critical service tier, the scale-up should complete within three hours.

### 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-AzureRmSqlDatabaseActivity` and `Stop-AzureRmSqlDatabaseActivity`.

- 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 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, upgrading the secondary database first is required.
- 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, downgrading the primary database first is required.
- The new properties for the database are not applied until the changes are complete.

## DTU-based purchasing model: Change storage size

- 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](#).

## DTU-based purchasing model: Change compute resources (DTUs)

After initially picking a service tier, compute size, and storage amount, 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.

Changing the service tier and/or compute size of a database creates a replica of the original database at the new compute size, and then switches connections over to the replica. No data is lost during this process but during the brief moment when we switch over to the replica, connections to the database are disabled, so some transactions in flight may be rolled back. The length of time for the switch-over varies, but is less than 30 seconds 99% of the time. If there are large numbers of transactions in flight at the moment connections are disabled, the length of time for the switch-over may be longer.

The duration of the entire scale-up process depends on both the size and service tier of the database before and after the change. For example, a 250-GB database that is changing to, from, or within a Standard service tier, should complete within six hours. For a database the same size that is changing compute sizes within the Premium service tier, the scale-up should complete within three hours.

**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-AzureRmSqlDatabaseActivity` and `Stop-AzureRmSqlDatabaseActivity`.

- 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 500 GB and its included storage amount is only 250 GB. So, the extra storage amount is  $500\text{ GB} - 250\text{ GB} = 250\text{ GB}$ . 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.
- 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, upgrading the secondary database first is required.
- 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, downgrading the primary database first is required.
- 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.

## DTU-based purchasing model: Limitations of P11 and P15 when the maximum size greater than 1 TB

A maximum size greater than 1 TB for P11 and P15 database is supported in the following regions: Australia East, Australia Southeast, Brazil South, Canada Central, Canada East, Central US, France Central, Germany Central, Japan East, Japan West, Korea Central, North Central US, North Europe, South Central US, South East Asia, UK South, UK West, US East2, West US, US Gov Virginia, and West Europe. The following considerations and limitations apply to P11 and P15 databases with a maximum size greater than 1 TB:

- If you choose a maximum size greater than 1 TB when creating a database (using a value of 4 TB or 4096 GB), the create command fails with an error if the database is provisioned in an unsupported region.
- For existing P11 and P15 databases located in one of the supported regions, you can increase the maximum storage to beyond 1 TB in increments of 256 GB up to 4 TB. To see if a larger size is supported in your region, use the [DATABASEPROPERTYEX](#) function or inspect the database size in the Azure portal. Upgrading an existing P11 or P15 database can only be performed by a server-level principal login or by members of the dbmanager database role.
- If an upgrade operation is executed in a supported region the configuration is updated immediately. The database remains online during the upgrade process. However, you cannot utilize the full amount of storage beyond 1 TB of storage until the actual database files have been upgraded to the new maximum size. The length of time required depends upon the size of the database being upgraded.
- When creating or updating a P11 or P15 database, you can only choose between 1-TB and 4-TB maximum size in increments of 256 GB. When creating a P11/P15, the default storage option of 1 TB is pre-selected. For databases located in one of the supported regions, you can increase the storage maximum to up to a maximum of 4 TB for a new or existing single database. For all other regions, the maximum size cannot be

increased above 1 TB. The price does not change when you select 4 TB of included storage.

- If the maximum size of a database is set to greater than 1 TB, then it cannot be changed to 1 TB even if the actual storage used is below 1 TB. Thus, you cannot downgrade a P11 or P15 with a maximum size larger than 1 TB to a 1 TB P11 or 1 TB P15 or lower compute size, such as P1-P6). This restriction also applies to the restore and copy scenarios including point-in-time, geo-restore, long-term-backup-retention, and database copy. Once a database is configured with a maximum size greater than 1 TB, all restore operations of this database must be run into a P11/P15 with a maximum size greater than 1 TB.
- 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](#).

# Elastic pools help you manage and scale multiple Azure SQL databases

10/30/2018 • 12 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 is 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](#).

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?

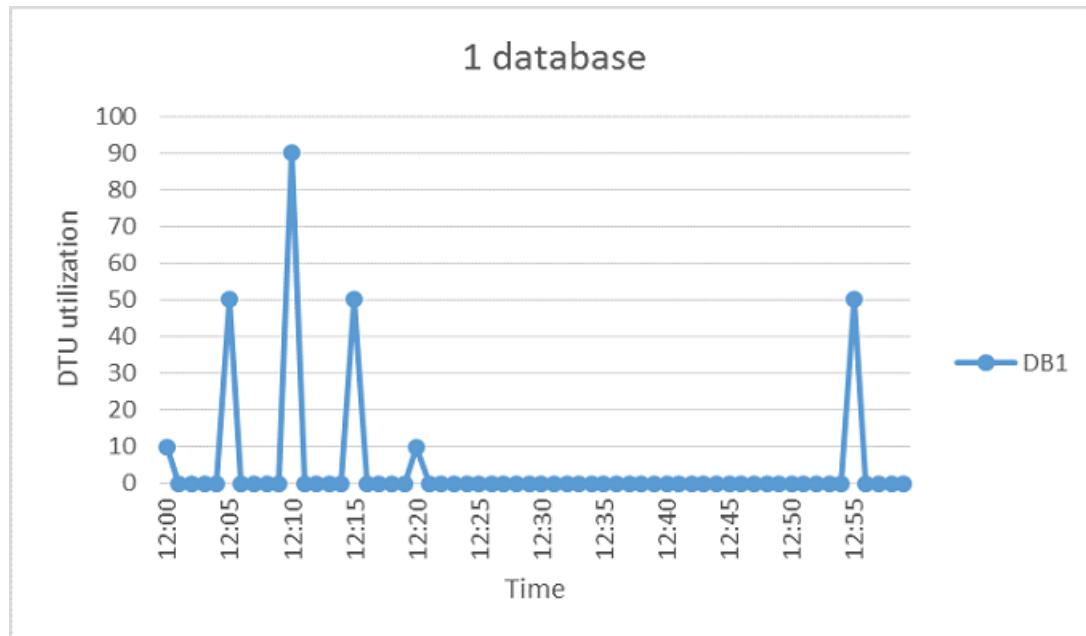
Pools are well suited for a large number of databases with specific utilization patterns. For a given database, this pattern is characterized by low average utilization with relatively infrequent utilization spikes.

The more databases you can add to a pool the greater your savings become. Depending on your application utilization pattern, it is 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

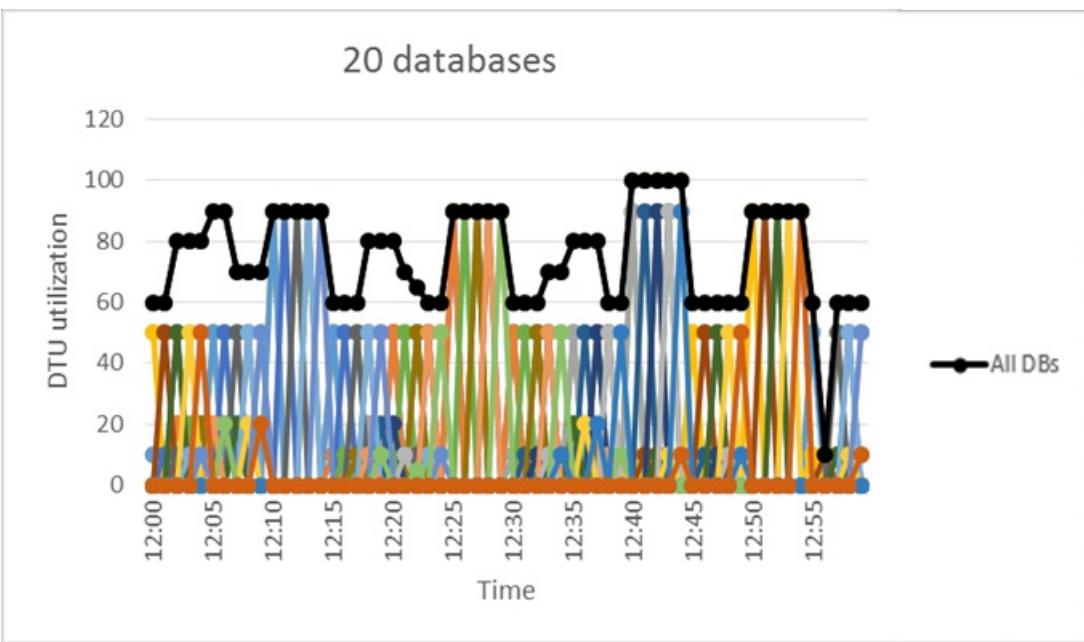
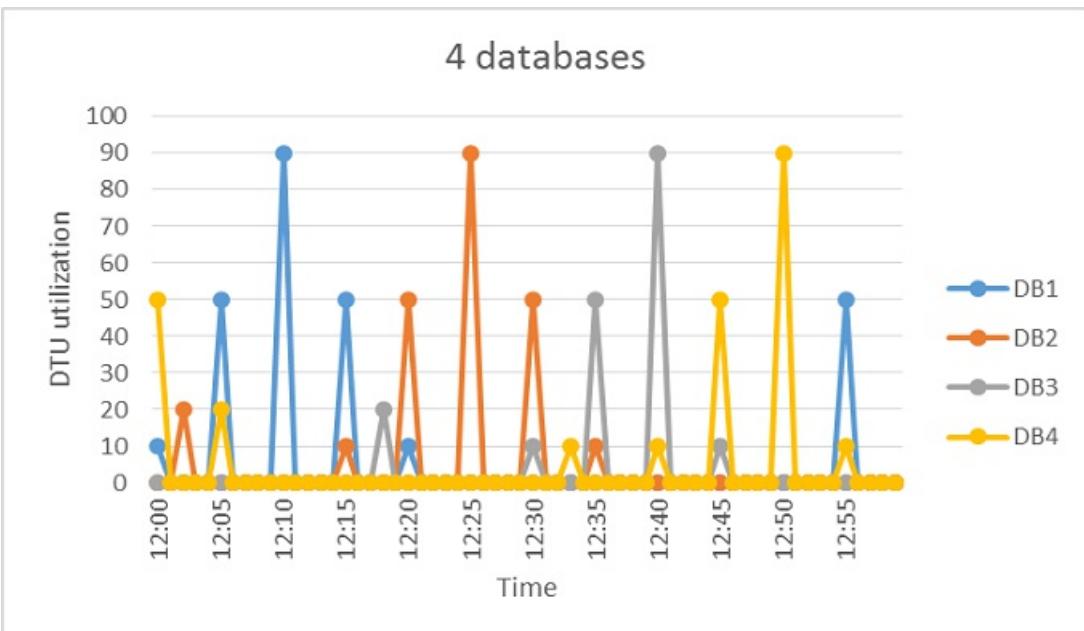
The following figure shows an example of a database that spends much time idle, but also periodically spikes with activity. This is a utilization pattern that is suited for a pool:



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 does not 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: MAX(<Total number of DBs X average DTU utilization per DB>, <Number of concurrently peaking DBs X Peak DTU utilization per DB>)

For vCore-based purchasing model: MAX(<Total number of DBs X average vCore utilization per DB>, <Number of concurrently peaking DBs X Peak vCore utilization per DB>)

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. To begin, see [Getting started with Elastic jobs](#).

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](#).

## 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. You can create an elastic pool by searching **SQL elastic pool** in the **Marketplace** or clicking **+Add** on the SQL elastic pools browse blade. You are able to specify a new or existing server through this pool provisioning workflow.
2. Or you can create an elastic pool by navigating to an existing SQL server and clicking **Create pool** to create a pool directly into that server. The only difference here is you skip the step where you specify the server during the pool provisioning workflow.

#### 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.

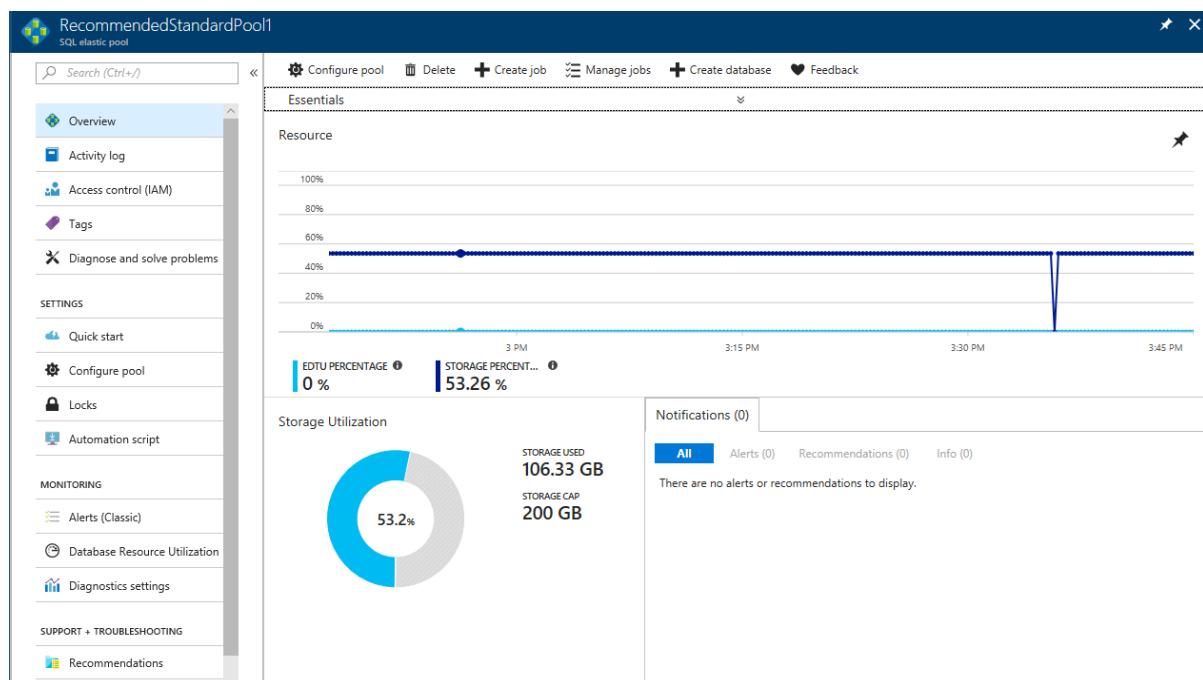
## Monitor an elastic pool and its databases

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 will first see a screen that gives you an overview of the status of your elastic pool. This includes:

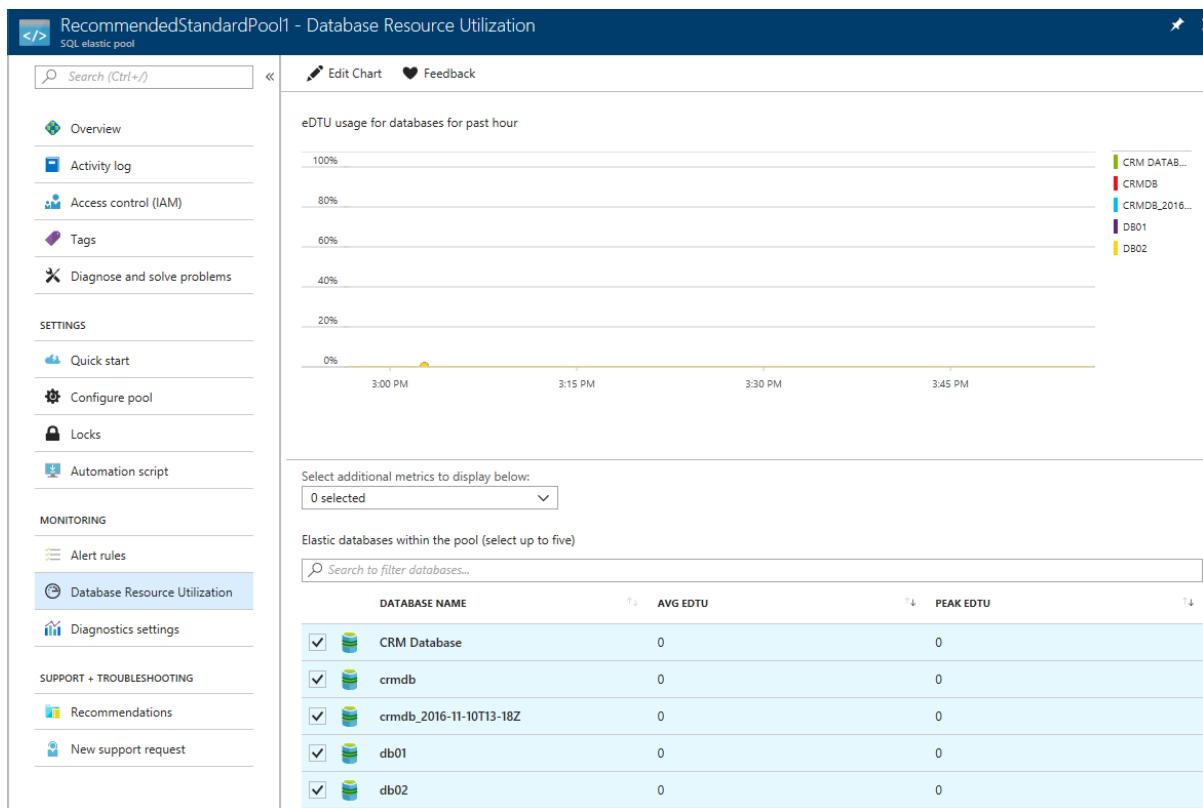
- Monitoring charts showing resources usage of the elastic pool
- Recent alerts and recommendations, if available, for the elastic pool

The following graphic shows an example elastic pool:



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.



## 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](#).

## 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](#).

# Create and manage elastic pools in Azure SQL Database

10/30/2018 • 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 navigation sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, and Configure pool (which is selected). The main content area has several sections:

- Service Tiers:** Shows 'General Purpose' (scalable compute and storage options for budget-oriented applications, 1-16 vCores, starting at 211.17 USD / month) and 'Business Critical' (for applications with high transaction rate and highly resilient to failures, 2-16 vCores, starting at 1103.55 USD / month).
- Databases:** A table showing nine databases (DB1 to DB9) with columns: DATABASE NAME, AVG CPU, PEAK CPU, and SIZE. All databases have 0 values in these columns.
- Cost Summary:** A table showing estimated costs for the selected service tier:

|                              |            |
|------------------------------|------------|
| 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

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 logical servers for an elastic pool, see [Create and Managed logical 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   |
|-----------------------------------|---|
| New-AzureRmSqlElasticPool         | Creates an elastic database pool on a logical SQL server.   |
| Get-AzureRmSqlElasticPool         | Gets elastic pools and their property values on a logical SQL server.   |
| Set-AzureRmSqlElasticPool         | Modifies properties of an elastic database pool on a logical SQL server. For example, use the <b>StorageMB</b> property to modify the max storage of an elastic pool. |
| Remove-AzureRmSqlElasticPool      | Deletes an elastic database pool on a logical SQL server.   |
| Get-AzureRmSqlElasticPoolActivity | Gets the status of operations on an elastic pool on a logical SQL server.   |
| New-AzureRmSqlDatabase            | Creates a new database in an existing pool or as a single database.   |
| Get-AzureRmSqlDatabase            | Gets one or more databases.   |
| Set-AzureRmSqlDatabase            | Sets properties for a database, or moves an existing database into, out of, or between elastic pools.   |
| Remove-AzureRmSqlDatabase         | 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                                  |
|----------------------------|--|
| az sql elastic-pool create | Creates an elastic pool.                     |
| az sql elastic-pool list   | Returns a list of elastic pools in a server. |

| CMDLET  | DESCRIPTION  |
|---|--|
| <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 database pools in a logical server. For each elastic database 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.                       |
| <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   |
|----------------------------------|---|
| Elastic pools - Create Or Update | Creates a new elastic pool or updates an existing elastic pool. |
| Elastic pools - Delete           | Deletes the elastic pool.                                       |
| Elastic pools - Get              | Gets an elastic pool.   |
| Elastic pools - List By Server   | Returns a list of elastic pools in a server.                    |
| Elastic pools - Update           | Updates an existing elastic pool.                               |
| Elastic pool Activities          | Returns elastic pool activities.                                |
| Elastic pool Database Activities | Returns activity on databases inside of an elastic pool.        |
| Databases - Create Or Update     | Creates a new database or updates an existing database.         |
| Databases - Get                  | Gets a database.  |
| Databases - List By Elastic Pool | Returns a list of databases in an elastic pool.                 |
| Databases - List By Server       | Returns a list of databases in a server.                        |
| Databases - Update               | Updates an existing database.                                   |

## Next steps

- 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](#).

# Azure SQL Database vCore-based purchasing model limits for elastic pools

10/16/2018 • 8 minutes to read • [Edit Online](#)

This article provides the detailed resource limits for Azure SQL Database elastic pools and pooled databases using the vCore-based purchasing model.

For DTU-based purchasing model limits, see [SQL Database DTU-based 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](#).

## **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.

## General Purpose service tier: Storage sizes and compute sizes

## Generation 4 compute platform

| <b>COMPUTE SIZE</b>                             | <b>GP_GEN4_1</b>                 | <b>GP_GEN4_2</b>                 | <b>GP_GEN4_4</b>                 | <b>GP_GEN4_8</b>                 | <b>GP_GEN4_16</b>                | <b>GP_GEN4_24</b>                |
|---|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| Max data size (GB)                              | 512                              | 756                              | 1536                             | 2048                             | 3584                             | 4096                             |
| Max log size                                    | 154                              | 227                              | 461                              | 614                              | 1075                             | 1229                             |
| TempDB size(DB)                                 | 32                               | 64                               | 128                              | 256                              | 384                              | 384                              |
| Target IOPS (64 KB)                             | 500                              | 1000                             | 2000                             | 4000                             | 7000                             | 7000                             |
| IO latency (approximate)                        | 5-7 ms (write)<br>5-10 ms (read) |
| Max concurrent workers (requests)               | 210                              | 420                              | 840                              | 1680                             | 3360                             | 5040                             |
| Max allowed sessions                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            |
| Max number DBs per pool                         | 100                              | 200                              | 500                              | 500                              | 500                              | 500                              |
| 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, 2, 4            | 0, 0.25, 0.5, 1, 2, 4, 8         | 0, 0.25, 0.5, 1, 2, 4, 8, 16     | 0, 0.25, 0.5, 1, 2, 4, 8, 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                       |

### Generation 5 compute platform

| <b>COMPUTE SIZE</b> | <b>GP_GEN5_2</b> | <b>GP_GEN5_4</b> | <b>GP_GEN5_8</b> | <b>GP_GEN5_16</b> | <b>GP_GEN5_24</b> | <b>GP_GEN5_32</b> | <b>GP_GEN5_40</b> | <b>GP_GEN5_80</b> |
|---------------------|------------------|------------------|------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| H/W generation      | 5                | 5                | 5                | 5                 | 5                 | 5                 | 5                 | 5                 |
| vCores              | 2                | 4                | 8                | 16                | 24                | 32                | 40                | 80                |

| <b>COMPUTE SIZE</b>               | <b>GP_GEN5_2</b>                 | <b>GP_GEN5_4</b>                 | <b>GP_GEN5_8</b>                 | <b>GP_GEN5_16</b>                | <b>GP_GEN5_24</b>                | <b>GP_GEN5_32</b>                | <b>GP_GEN5_40</b>                | <b>GP_GEN5_80</b>                |
|-----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| Memory (GB)                       | 11                               | 22                               | 44                               | 88                               | 132                              | 176                              | 220                              | 440                              |
| Columnstore support               | Yes                              |
| In-memory OLTP storage (GB)       | N/A                              |
| Storage type                      | Premium (Remote) Storage         |
| Max data size (GB)                | 512                              | 756                              | 1536                             | 2048                             | 3072                             | 4096                             | 4096                             | 4096                             |
| Max log size                      | 154                              | 227                              | 461                              | 614                              | 922                              | 1229                             | 1229                             | 1229                             |
| TempDB size(DB)                   | 64                               | 128                              | 256                              | 384                              | 384                              | 384                              | 384                              | 384                              |
| Target IOPS (64 KB)               | 500                              | 1000                             | 2000                             | 4000                             | 6000                             | 7000                             | 7000                             | 7000                             |
| IO latency (approximate)          | 5-7 ms (write)<br>5-10 ms (read) |
| Max concurrent workers (requests) | 210                              | 420                              | 840                              | 1680                             | 2520                             | 3360                             | 4200                             | 8400                             |
| Max allowed sessions              | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            | 30000                            |
| Max number DBs per pool           | 100                              | 200                              | 500                              | 500                              | 500                              | 500                              | 500                              | 500                              |

| <b>COMPUTE SIZE</b>                             | <b>GP_GEN5_2</b>   | <b>GP_GEN5_4</b>      | <b>GP_GEN5_8</b>         | <b>GP_GEN5_16</b>            | <b>GP_GEN5_24</b>                | <b>GP_GEN5_32</b>              | <b>GP_GEN5_40</b>                  | <b>GP_GEN5_80</b>                      |
|---|--------------------|-----------------------|--------------------------|------------------------------|----------------------------------|--------------------------------|------------------------------------|--|
| Min/max elastic pool vCore choices per database | 0, 0.25, 0.5, 1, 2 | 0, 0.25, 0.5, 1, 2, 4 | 0, 0.25, 0.5, 1, 2, 4, 8 | 0, 0.25, 0.5, 1, 2, 4, 8, 16 | 0, 0.25, 0.5, 1, 2, 4, 8, 16, 24 | 0, 0.5, 1, 2, 4, 8, 16, 24, 32 | 0, 0.5, 1, 2, 4, 8, 16, 24, 32, 40 | 0, 0.5, 1, 2, 4, 8, 16, 24, 32, 40, 80 |
| Number of replicas                              | 1                  | 1                     | 1                        | 1                            | 1                                | 1                              | 1                                  | 1                                      |
| Multi-AZ  | N/A                | 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                                | 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                         | 1X DB size                             |
|   |                    |                       |                          |                              |                                  |                                |                                    |  |

## Business Critical service tier: Storage sizes and compute sizes

### Generation 4 compute platform

| <b>COMPUTE SIZE</b>         | <b>BC_GEN4_1</b> | <b>BC_GEN4_2</b> | <b>BC_GEN4_4</b> | <b>BC_GEN4_8</b> | <b>BC_GEN4_16</b> | <b>BC_GEN4_24</b> |
|-----------------------------|------------------|------------------|------------------|------------------|-------------------|-------------------|
| H/W generation              | 4                | 4                | 4                | 4                | 4                 | 4                 |
| vCores                      | 1                | 2                | 4                | 8                | 16                | 24                |
| Memory (GB)                 | 7                | 14               | 28               | 56               | 112               | 168               |
| Columnstore support         | Yes              | Yes              | Yes              | Yes              | Yes               | Yes               |
| In-memory OLTP storage (GB) | 1                | 2                | 4                | 8                | 20                | 36                |
| Storage type                | Local SSD         | Local SSD         |
| Max data size (GB)          | 1024             | 1024             | 1024             | 1024             | 1024              | 1024              |
| Max log size                | 307              | 307              | 307              | 307              | 307               | 307               |
| TempDB size(DB)             | 32               | 64               | 128              | 256              | 384               | 384               |

## **Generation 5 compute platform**



| <b>COMPUTE SIZE</b>     | <b>BC_GEN5_2</b> | <b>BC_GEN5_4</b> | <b>BC_GEN5_8</b> | <b>BC_GEN5_16</b> | <b>BC_GEN5_24</b> | <b>BC_GEN5_32</b> | <b>BC_GEN5_40</b> | <b>BC_GEN5_80</b> |
|-------------------------|------------------|------------------|------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| Read Scale-out          | Yes              | 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        | 1X DB size        |

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.

| <b>PROPERTY</b>          | <b>DESCRIPTION</b>   |
|--------------------------|--|
| 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

- See [SQL Database FAQ](#) for answers to frequently asked questions.
- See [Overview of resource limits on a logical server](#) for information about limits at the server and subscription levels.
- For information about general Azure limits, see [Azure subscription and service limits, quotas, and constraints](#).

# Resources limits for elastic pools using the DTU-based purchasing model

10/5/2018 • 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-based purchasing model.

For DTU-based purchasing model resource limits for single databases, see [DTU-based resource limits - single databases](#). For vCore-based resource limits, see [vCore-based resource limits - single databases](#) and [vCore-based 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](#).

## 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](#).

## 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                               |
| Max number DBs per pool                  | 100          | 200                | 500                      | 500                       | 500                             | 500                               |

| <b>EDTUS PER POOL</b>                      | <b>50</b>     | <b>100</b>         | <b>200</b>              | <b>300</b>                   | <b>400</b>                        | <b>800</b>                             |
|--|---------------|--------------------|-------------------------|------------------------------|-----------------------------------|--|
| 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, 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, 1500 |
| 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, 1500    |
| Max storage per database (GB)              | 1024                | 1024                    | 1024                         | 1024                               | 1024                                     |
|  |                     |                         |                              |                                    |  |

### Premium elastic pool limits (continued)

| EDTUS PER POOL                             | 2000                                     | 2500                                     | 3000                                     | 3500                                     | 4000   |
|--|--|--|--|--|--|
| 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 the following: West Central US, China East, USDoDCentral, Germany Central, USDoDEast, US Gov SouthWest, Germany NorthEast, USGov Iowa, China North. In other regions, the storage max in the Premium tier is limited to 1 TB. 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.

### Database properties for pooled databases

The following table describes the properties for pooled databases.

| PROPERTY | DESCRIPTION |
|----------|-------------|
|----------|-------------|

| 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

- See [SQL Database FAQ](#) for answers to frequently asked questions.
- See [Overview of resource limits on a logical server](#) for information about limits at the server and subscription levels.
- 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 <https://docs.microsoft.com/sql/relational-databases/databases/tempdb-database#tempdb-database-in-sql-database>.

# Scale elastic pool resources in Azure SQL Database

10/30/2018 • 4 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.

## vCore-based purchasing model: Change elastic pool storage size

- Storage can be provisioned up to the max size limit:
  - For Standard storage, increase or decrease size in 10-GB increments
  - For Premium storage, 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](#).

## vCore-based purchasing model: Change elastic pool compute resources (vCores)

You can increase or decrease the compute size to an elastic pool based on resource needs using the [Azure portal](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

- When rescaling vCores in an elastic pool, database connections are briefly dropped. This behavior is the same behavior that occurs when rescaling DTUs for a single database. For details on the duration and impact of dropped connections for a database during rescaling operations, see [Change compute resources \(DTUs\)](#).
- The duration to rescale pool vCores can depend on the total amount of storage space used by all databases in the pool. In general, the rescaling latency averages 90 minutes or less per 100 GB. For example, if the total space used by all databases in the pool is 200 GB, then the expected latency for rescaling the pool is 3 hours or less. In some cases within the Standard or Basic tier, the rescaling latency can be under five minutes regardless of the amount of space used.
- In general, the duration to change the min vCores per database or max vCores per database is five minutes or less.
- When downsizing pool vCores, the pool used space must be smaller than the maximum allowed size of the target service tier and pool vCores.

## DTU-based purchasing model: Change elastic pool storage size

- 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](#).

## DTU-based purchasing model: Change elastic pool compute resources (eDTUs)

You can increase or decrease the resources available to an elastic pool based on resource needs using the [Azure portal](#), [PowerShell](#), the [Azure CLI](#), or the [REST API](#).

- When rescaling pool eDTUs, database connections are briefly dropped. This behavior is the same behavior that occurs when rescaling DTUs for a single database. For details on the duration and impact of dropped connections for a database during rescaling operations, see [Change compute resources \(DTUs\)](#).
- The duration to rescale pool eDTUs can depend on the total amount of storage space used by all databases in the pool. In general, the rescaling latency averages 90 minutes or less per 100 GB. For example, if the total space used by all databases in the pool is 200 GB, then the expected latency for rescaling the pool is 3 hours or less. In some cases within the Standard or Basic tier, the rescaling latency can be under five minutes regardless of the amount of space used.
- In general, the duration to change the min eDTUs per database or max eDTUs per database is five minutes or less.
- When downsizing 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.

## Next steps

For overall resource limits, see [SQL Database vCore-based resource limits - elastic pools](#) and [SQL Database DTU-based resource limits - elastic pools](#).

# Use SQL Database Managed Instance with virtual networks and near 100% compatibility

10/26/2018 • 14 minutes to read • [Edit Online](#)

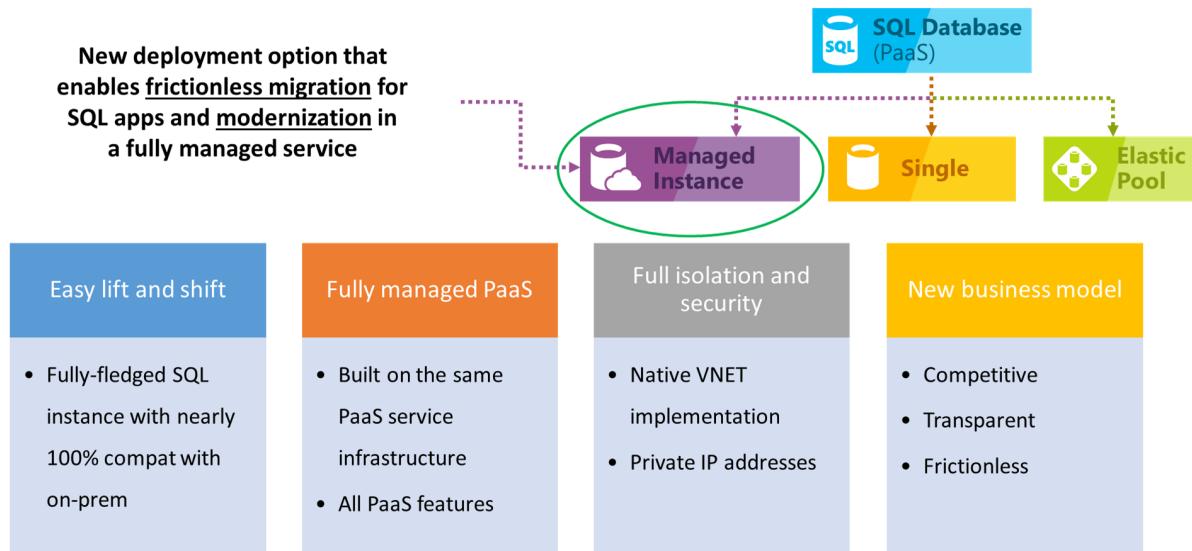
Azure SQL Database Managed Instance is a new deployment model of Azure SQL Database, providing near 100% compatibility with the latest SQL Server on-premises (Enterprise Edition) Database Engine, providing a native [virtual network \(VNet\)](#) implementation that addresses common security concerns, and a [business model](#) favorable for on-premises SQL Server customers. Managed Instance 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, Managed Instance 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 Managed Instance is currently available, see [supported regions](#).

The following diagram outlines key features of the Managed Instance:

## What is SQL Database Managed Instance?



Azure SQL Database Managed Instance 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 their existing licenses for discounted rates on a SQL Database Managed Instance using the [Azure Hybrid Benefit for SQL Server](#). SQL Database Managed Instance is the best migration destination in the cloud for SQL Server instances that require high security and a rich programmability surface.

By General Availability, Managed Instance 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 Azure SQL Database Single Database, Azure SQL Database Managed Instance, and SQL Server IaaS hosted in Virtual Machine see [how to choose the right version of SQL Server in Azure cloud](#).

## Key features and capabilities

Azure SQL Database 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<br>No management overhead for managing underlying infrastructure<br>Quick provisioning and service scaling<br>Automated patching and version upgrade<br>Integration with other PaaS data services  | 99.99% uptime SLA<br>Built in <a href="#">high-availability</a><br>Data protected with <a href="#">automated backups</a><br>Customer configurable backup retention period (fixed to 7 days in Public Preview)<br>User-initiated <a href="#">backups</a><br><a href="#">Point in time database restore</a> capability |
| <b>Security and compliance</b><br><br>Isolated environment ( <a href="#">VNet integration</a> , single tenant service, dedicated compute and storage)<br><a href="#">Transparent data encryption (TDE)</a><br><a href="#">Azure AD authentication</a> , single sign-on support<br>Adheres to compliance standards same as Azure SQL database<br><a href="#">SQL auditing</a><br><a href="#">Threat Detection</a> | <b>Management</b><br><br>Azure Resource Manager API for automating service provisioning and scaling<br>Azure portal functionality for manual service provisioning and scaling<br>Data Migration Service  |

The key features of Managed Instance 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  |
| Number of data files (ROWS) per the database          | Multiple                                   |
| Number of log files (LOG) per database                | 1  |
| VNet - Azure Resource Manager deployment              | Yes  |

| FEATURE                             | DESCRIPTION  |
|-------------------------------------|--|
| 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](#) in Managed Instance 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 30 percent savings with the [Azure Hybrid Benefit for SQL Server](#).

In vCore model, you can choose between generations of hardware.

- **Gen 4** Logical CPUs are based on Intel E5-2673 v3 (Haswell) 2.4-GHz processors, attached SSD, physical cores, 7GB RAM per core, and compute sizes between 8 and 24 vCores.
- **Gen 5** Logical CPUs are based on Intel E5-2673 v4 (Broadwell) 2.3-GHz processors, fast eNVM SSD, hyper-threaded logical core, and compute sizes between 8 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 (preview)**: 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](#).

### IMPORTANT

Changing your service tier from General Purpose to Business Critical or vice versa is not supported in Public Preview. If you want to migrate your databases to an instance in different service tier, you can create new instance, restore databases with point in time restore from the original instance and then drop original instance if it is not needed anymore. However, you can scale your number of vCores and storage up or down within a service tier with no downtime.

### 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 Premium storage (8 TB)
- Built-in [High-availability](#) based on reliable Azure Premium Storage and [Azure Service Fabric](#)

For more information, see [Storage layer in General purpose tier](#) and [Storage performance best practices and considerations for Azure SQL DB Managed Instance \(General Purpose\)](#).

Find more information about the difference between service tiers in [Managed Instance resource limits](#).

### **Business Critical service tier (preview)**

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 SSD storage (up to 1 TB on Gen 4 and up to 4 TB on Gen 5)
- 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

#### **IMPORTANT**

The **Business Critical** service tier is in preview.

Find more information about the difference between service tiers in [Managed Instance resource limits](#).

## **Advanced security and compliance**

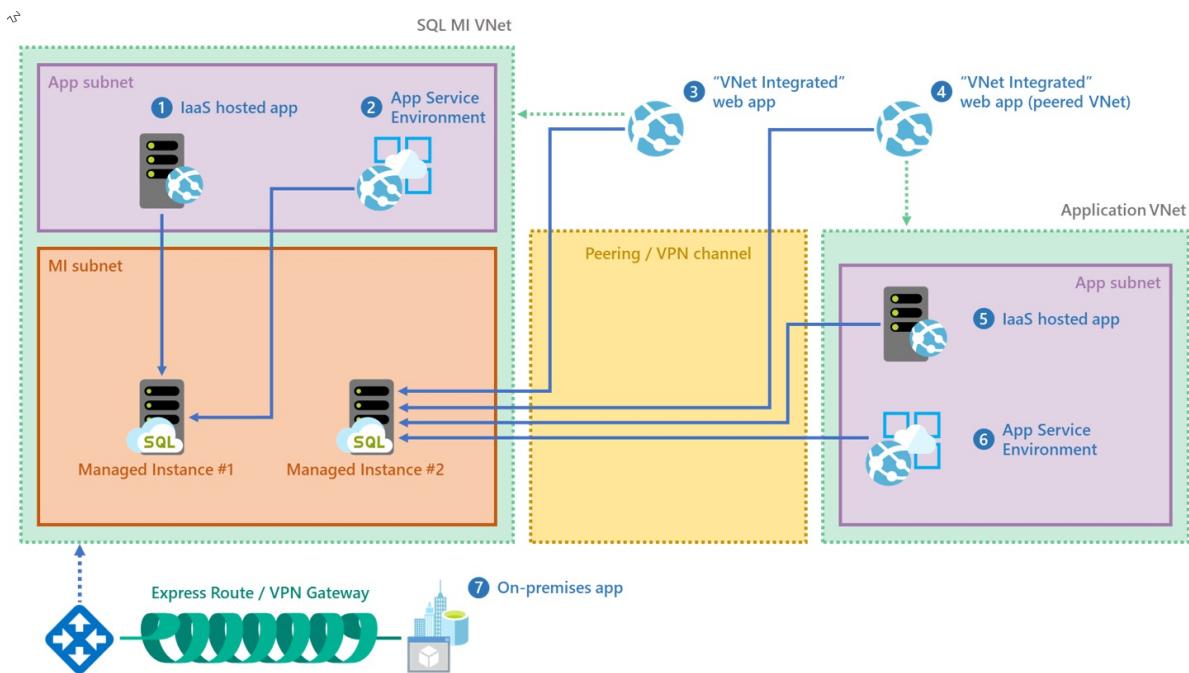
Azure SQL Database Managed Instance combines advanced security features provided by Azure cloud and SQL Server Database Engine.

### **Managed Instance security isolation in Azure cloud**

Managed Instance provide 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
- 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 [Configure a VNet for Azure SQL Database Managed Instance](#) and [Connect your application to Azure SQL Database 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 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 - Managed Instance secures your data by providing encryption for data in motion using Transport Layer Security. In addition to transport layer security, SQL Database Managed Instance 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.
- **Threat Detection** complements **Managed Instance 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. Threat Detection 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 Azure SQL Database 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 Managed Instance with transparent data encryption. TDE is SQL's proven encryption-at-rest technology that is required by many compliance standards to protect against theft of storage media. During public preview, the automatic key management model is supported (performed by the PaaS platform).

Migration of an encrypted database to SQL Managed Instance is supported via the Azure Database Migration Service (DMS) or native restore. If you plan to migrate encrypted database using native restore, migration of the existing TDE certificate from the SQL Server on-premise or SQL Server VM to Managed instance is a required step. For more information about migration options, see [SQL Server instance migration to Azure SQL Database Managed Instance](#).

## Azure Active Directory Integration

Azure SQL Database Managed Instance supports traditional SQL server Database engine logins and logins integrated with Azure Active Directory (AAD). AAD Logins are Azure cloud version of Windows database logins that you are using in your on-premises environment.

### Azure Active Directory integration and multi-factor authentication

Managed Instance 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. Managed Instance has same authorization capabilities as SQL Server 2017.

## Database migration

Managed Instance targets user scenarios with mass database migration from on-premises or IaaS database

implementations. Managed Instance supports several database migration options:

### Backup and restore

The migration approach leverages SQL backups to Azure blob storage. Backups stored in Azure storage blob can be directly restored into 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 an Azure SQL Database logical server single or pooled database.

### 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. Deployment options include Azure SQL Database, Managed Instance, and SQL Server in Azure VM at Public Preview. See [How to migrate your on-premises database to Managed Instance using DMS](#).

## SQL features supported

Managed Instance 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](#).

Managed Instance 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> |

Note: Features will be added in stages until General Availability of Managed Instance.

### Key differences between SQL Server on-premises and Managed Instance

Managed Instance 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 service is not 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 and Managed Instance, see [Compare SQL Database logical server and Managed Instance](#).

## Managed Instance administration features

Managed Instance enable system administrator to focus on what matters the most for business. Many system administrator/DBA activities are not required, or they are simple. 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.           |
| <code>SERVERPROPERTY ('Edition')</code>      | SQL Azure  | This value is same as in SQL Database.           |
| <code>SERVERPROPERTY('EngineEdition')</code> | 8  | This value uniquely identifies Managed Instance. |

| PROPERTY  | VALUE  | COMMENT   |
|---|--|---|
| <code>@@SERVERNAME ,<br/>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 auto-generated 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 [Quick-start 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](#).

# Overview Azure SQL Database Managed Instance resource limits

10/17/2018 • 5 minutes to read • [Edit Online](#)

This article provides an overview of the Azure SQL Database Managed Instance resource limits and provides information how to create request to increase default regional subscription limits.

## NOTE

For other Managed Instance limitations, see [vCore-based purchasing model](#) and [Managed Instance service tiers](#). For differences in supported features and T-SQL statements see [Feature differences](#) and [T-SQL statement support](#).

## Instance-level resource limits

Managed Instance has characteristics and resource limits that depends on the underlying infrastructure and architecture. Limits depend on hardware generation and service tier.

### Hardware generation characteristics

Azure SQL Database Managed Instance can be deployed on two hardware generation (Gen4 and Gen5).

Hardware generations have different characteristics that are described in the following table:

|                                 | GEN 4  | GEN 5   |
|---------------------------------|--|---|
| 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 processors, fast eNVM SSD, vCore=1 LP (hyper-thread) |
| Compute                         | 8, 16, 24 vCores   | 8, 16, 24, 32, 40, 64, 80 vCores  |
| Memory                          | 7 GB per vCore   | 5.1 GB per vCore  |
| Max storage (Business Critical) | 1 TB   | 1 TB, 2 TB, or 4 TB depending on the number of cores                                      |

### Service tier characteristics

Managed Instance has two service tiers - General Purpose and Business Critical (Public Preview). These tiers provide different capabilities, as described in the table below:

| FEATURE           | GENERAL PURPOSE   | BUSINESS CRITICAL (PREVIEW)   |
|-------------------|---|---|
| Number of vCores* | Gen4: 8, 16, 24<br>Gen5: 8, 16, 24, 32, 40, 64, 80                            | Gen4: 8, 16, 24, 32<br>Gen5: 8, 16, 24, 32, 40, 64, 80                        |
| Memory            | Gen4: 56GB-156GB<br>Gen5: 44GB-440GB<br>*Proportional to the number of vCores | Gen4: 56GB-156GB<br>Gen5: 44GB-440GB<br>*Proportional to the number of vCores |

| FEATURE                              | GENERAL PURPOSE   | BUSINESS CRITICAL (PREVIEW)  |
|--------------------------------------|---|--|
| Max storage size                     | 8 TB  | Gen 4: 1 TB<br>Gen 5:<br>- 1 TB for 8, 16 vCores<br>- 2 TB for 24 vCores<br>- 4 TB for 32, 40, 64, 80 vCores |
| Max storage per database             | Determined by the max storage size per instance         | Determined by the max storage size per instance  |
| Max number of databases per instance | 100   | 100  |
| Max database files per instance      | Up to 280   | Unlimited  |
| Expected max storage IOPS            | 500-5000 ( <a href="#">depends on data file size</a> ). | Depends on the underlying SSD speed.   |

## Supported regions

Managed Instances can be created only in [supported regions](#). If you want to create a Managed Instance in the region that is currently not supported, you can [send support request via 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 \(CSP\)](#)

### NOTE

This limitation is temporary. New subscription types will be enabled in the future.

## Regional resource limitations

Supported subscription types can contain a limited number of resources per region. Managed Instance has two default limits per Azure region depending on a type of subscription type:

- **Subnet limit:** The maximum number of subnets where managed instances are deployed in a single region.
- **Instance number limit:** The maximum number of instances that can be deployed in a single region.

In the following table are shown default regional limits for supported subscriptions:

| SUBSCRIPTION TYPE | MAX NUMBER OF MANAGED INSTANCE SUBNETS | MAX NUMBER OF INSTANCES | MAX NUMBER OF GP MANAGED INSTANCES* | MAX NUMBER OF BC MANAGED INSTANCES* |
|-------------------|--|-------------------------|-------------------------------------|-------------------------------------|
| Pay-as-you-go     | 1*                                     | 4*                      | 4*                                  | 1*                                  |
| CSP               | 1*                                     | 4*                      | 4*                                  | 1*                                  |
| EA                | 3**                                    | 12**                    | 12**                                | 3**                                 |

\* You can either deploy 1 BC or 4 GP instances in one subnet, so that total number of "instance units" in the subnet never exceeds 4.

\*\* Maximum number of instances in one service tier applies if there are no instances in another service tier. In case you plan to mix GP and BC instances within same subnet, use the following section as a reference for allowed combinations. As a simple rule, the total number of subnets cannot exceed 3, and the total number of instance units cannot exceed 12.

These limits can be increased by creating 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.

#### IMPORTANT

When planning your deployments, consider that a Business Critical (BC) instance (due to added redundancy) generally consumes 4x more capacity than a General Purpose (GP) instance. So, for your calculations, 1 GP instance = 1 instance unit and 1 BC instance = 4 instance units. To simplify your consumption analysis against the default limits, summarize the instance 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.

## Strategies for deploying mixed General Purpose and Business Critical instances

[Enterprise Agreement \(EA\)](#) subscriptions can have combinations of GP and BC instances. However, there are some constraints regarding the placement of the instances in the subnets.

#### NOTE

[Pay-as-you-go](#) and [Cloud Service Provider \(CSP\)](#) subscription types can have either one Business Critical or up to 4 General Purpose instances.

The following examples cover deployment cases with non-empty subnets and mixed GP and BC service tiers.

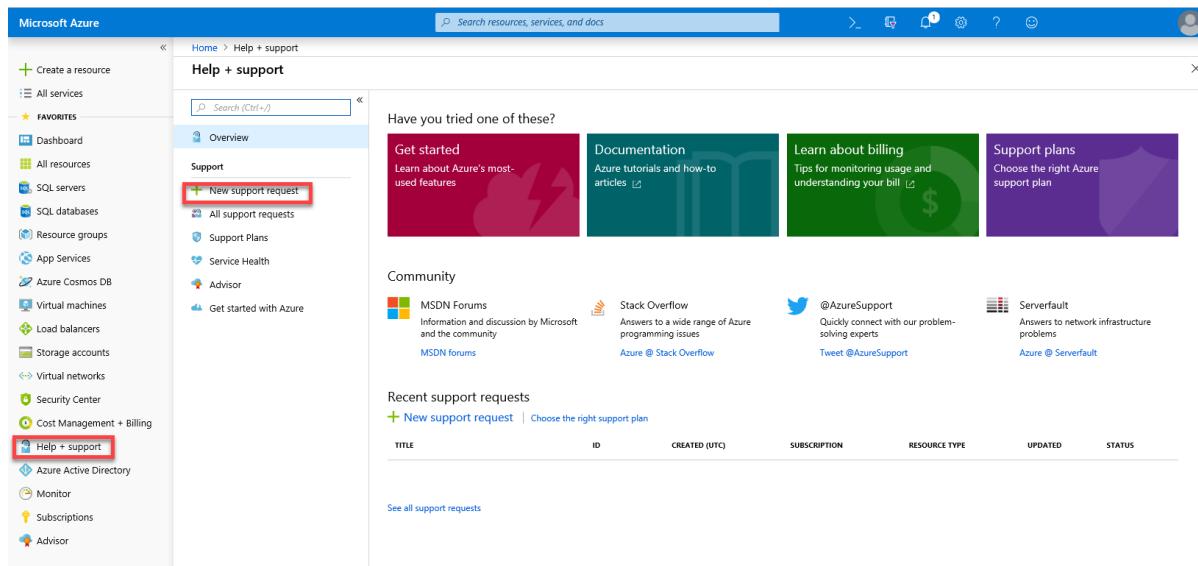
| NUMBER OF SUBNETS | SUBNET 1                                   | SUBNET 2                             | SUBNET 3         |
|-------------------|--|--------------------------------------|------------------|
| 1                 | 1 BC and up to 8 GP<br>2 BC and up to 4 GP | N/A                                  | N/A              |
| 2                 | 0 BC, up to 4 GP                           | 1 BC, up to 4 GP<br>2 BC, 0 GP       | N/A              |
| 2                 | 1 BC, 0 GP                                 | 0 BC, up to 8 GP<br>1 BC, up to 4 GP | N/A              |
| 2                 | 2 BC, 0 GP                                 | 0 BC, up to 4 GP                     | N/A              |
| 3                 | 1 BC, 0 GP                                 | 1 BC, 0 GP                           | 0 BC, up to 4 GP |
| 3                 | 1 BC, 0 GP                                 | 0 BC, up to 4 GP                     | 0 BC, up to 4 GP |

## Obtaining a larger quota for SQL Managed Instance

If you need more Managed Instances in your current regions, you can send the support request to extend the

quota using Azure portal. To initiate the process of obtaining a larger quota:

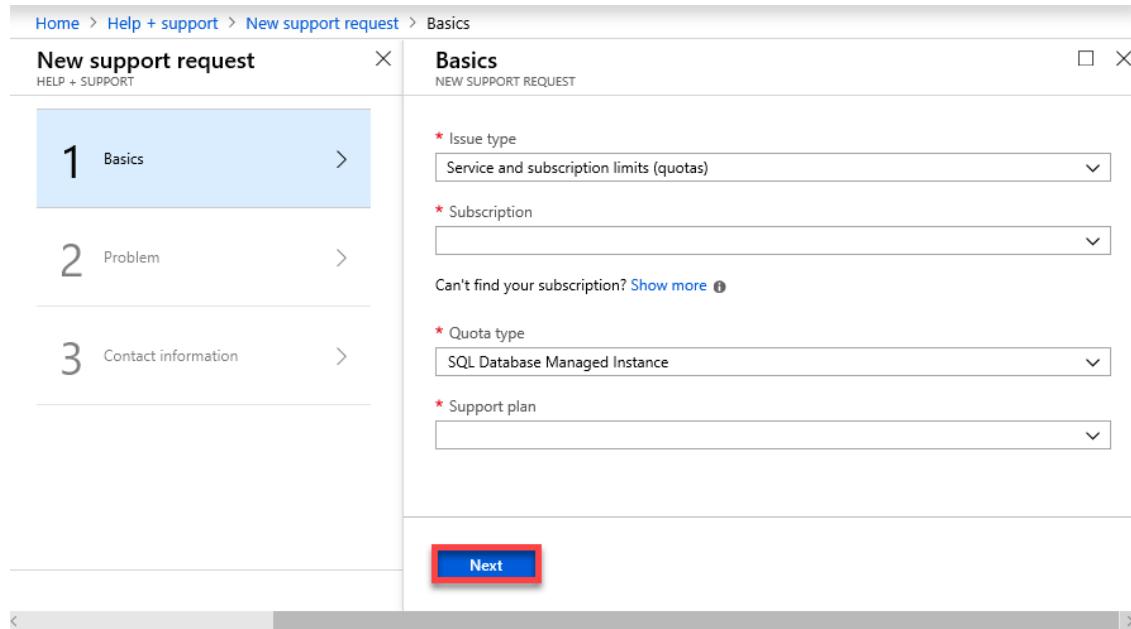
1. Open **Help + support**, and click **New support request**.



The screenshot shows the Microsoft Azure portal's 'Help + support' section. On the left, there's a sidebar with various service links like 'Dashboard', 'All resources', and 'Help + support'. The 'Help + support' link is also highlighted with a red box. In the main content area, there's a 'Get started' section with a lightning bolt icon, 'Documentation' with a book icon, 'Learn about billing' with a dollar sign icon, and 'Support plans' with a shield icon. Below these are sections for 'Community' (MSDN Forums, Stack Overflow, Azure Support Twitter), 'Recent support requests', and a table for viewing support requests. A red box highlights the '+ New support request' button in the sidebar.

2. On the Basics tab for the new support request:

- For **Issue type**, select **Service and subscription limits (quotas)**.
- For **Subscription**, select your subscription.
- For **Quota type**, select **SQL Database Managed Instance**.
- For **Support plan**, select your support plan.



The screenshot shows the 'New support request' wizard. The left sidebar lists three tabs: '1 Basics', '2 Problem', and '3 Contact information'. The 'Basics' tab is active and shown in detail on the right. It has four required fields with red asterisks: 'Issue type' (set to 'Service and subscription limits (quotas)'), 'Subscription' (a dropdown menu), 'Quota type' (set to 'SQL Database Managed Instance'), and 'Support plan' (a dropdown menu). A 'Next' button is at the bottom of the form. Red boxes highlight the 'Issue type', 'Subscription', 'Quota type', and 'Support plan' fields.

3. Click **Next**.

4. On the Problem tab for the new support request:

- For **Severity**, select the severity level of the problem.
- For **Details**, provide additional information about your issue, including error messages.
- For **File upload**, attach a file with more information (up to 4 MB).

The screenshot shows the 'New support request' wizard in the Microsoft Azure portal. The current step is 'Problem'. The left sidebar lists three steps: 1 Basics (completed), 2 Problem (selected), and 3 Contact information. The main area contains fields for 'Severity' (set to 'B - Moderate impact'), 'Details' (describing a quota request for West US region), and a 'File upload' section.

#### IMPORTANT

A valid request should include:

- Region in which subscription limit needs to be increased
- Required number of instances, per service tier in existing subnets after the quota increase (if any of the existing subnets needs to be expanded)
- Required number of new subnets and total number of instances per service tier within the new subnets (if you need to deploy managed instances in new subnets).

- Click **Next**.
- On the Contact Information tab for the new support request, enter preferred contact method (email or phone) and the contact details.
- Click **Create**.

## 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 [Quick-start guide](#).

# Configure a VNet for Azure SQL Database Managed Instance

9/25/2018 • 6 minutes to read • [Edit Online](#)

Azure SQL Database Managed Instance must be deployed within an Azure [virtual network \(VNet\)](#). This deployment enables the following scenarios:

- 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

## Plan

Plan how you deploy a Managed Instance in virtual network using your answers to the following questions:

- Do you plan to deploy single or multiple Managed Instances?

The number of Managed Instances determines the minimum size of the subnet to allocate for your Managed Instances. For more information, see [Determine the size of subnet for Managed Instance](#).

- Do you need to deploy your Managed Instance into an existing virtual network or you are creating a new network?

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 existing virtual network for Managed Instance](#).

If you plan to create new virtual network, see [Create new virtual network for Managed Instance](#).

## Requirements

To create a Managed Instance, create a dedicated subnet (the Managed Instance subnet) inside the virtual network that conforms to the following requirements:

- **Dedicated subnet:** The Managed Instance subnet must not contain any other cloud service associated with it, and it must not be a Gateway subnet. You won't be able to create a Managed Instance in a subnet that contains resources other than Managed Instance, and you can not later add other resources in the subnet.
- **Compatible Network Security Group (NSG):** An NSG associated with a Managed Instance subnet must contain rules shown in the following tables (Mandatory inbound security rules and Mandatory outbound security rules) in front of any other rules. You can use an NSG to fully control access to the Managed Instance data endpoint by filtering traffic on port 1433.
- **Compatible user-defined route table (UDR):** The Managed Instance subnet must have a user route table with **0.0.0.0/0 Next Hop Internet** as the mandatory UDR assigned to it. In addition, you can add a UDR that routes traffic that has on-premises private IP ranges as a destination through virtual network gateway or virtual network appliance (NVA).
- **Optional custom DNS:** If a custom DNS is specified on the virtual network, Azure's recursive resolver IP address (such as 168.63.129.16) must be added to the list. For more information, see [Configuring Custom DNS](#). The custom DNS server must be able to resolve host names in the following domains and their subdomains: *microsoft.com, windows.net, windows.com, msocsp.com, digicert.com, live.com, microsoftonline.com, and microsoftonline-p.com*.

- **No service endpoints:** The Managed Instance subnet must not have a service endpoint associated to it. Make sure that service endpoints option is disabled when creating the virtual network.
- **Sufficient IP addresses:** The Managed Instance subnet must have the bare minimum of 16 IP addresses (recommended minimum is 32 IP addresses). For more information, see [Determine the size of subnet for Managed Instances](#)

#### IMPORTANT

You won't be able to deploy a new Managed Instance if the destination subnet is not compatible with all of these requirements. When a Managed Instance is created, a *Network Intent Policy* is applied on the subnet to prevent non-compliant changes to networking configuration. After the last instance is removed from the subnet, the *Network Intent Policy* is removed as well

#### Mandatory inbound security rules

| NAME         | PORT                         | PROTOCOL | SOURCE            | DESTINATION | ACTION |
|--------------|------------------------------|----------|-------------------|-------------|--------|
| management   | 9000, 9003, 1438, 1440, 1452 | TCP      | Any               | Any         | Allow  |
| mi_subnet    | Any                          | Any      | MI SUBNET         | Any         | Allow  |
| health_probe | Any                          | Any      | AzureLoadBalancer | Any         | Allow  |

#### Mandatory outbound security rules

| NAME       | PORT           | PROTOCOL | SOURCE | DESTINATION | ACTION |
|------------|----------------|----------|--------|-------------|--------|
| management | 80, 443, 12000 | TCP      | Any    | Any         | Allow  |
| mi_subnet  | Any            | Any      | Any    | MI SUBNET   | Allow  |

## Determine the size of subnet for Managed Instances

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 subnet masks /28 to /24 when defining your subnet IP ranges.

#### IMPORTANT

Subnet size with 16 IP addresses is the bare minimum with limited potential for the further Managed Instance scale out. Choosing subnet with the prefix /27 or below is highly recommended.

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.

## Create a new virtual network for a Managed Instance

The easiest way to create and configure virtual network is to use Azure Resource Manager deployment template.

1. Sign in to the Azure portal.
2. Use **Deploy to Azure** button to deploy virtual network in Azure cloud:



This button will open a form that you can use to configure network environment where you can deploy Managed Instance.

#### NOTE

This Azure Resource Manager template will deploy virtual network with two subnets. One subnet called **ManagedInstances** is reserved for Managed Instances and has pre-configured route table, while the other subnet called **Default** is used for other resources that should access Managed Instance (for example, Azure Virtual Machines). You can remove **Default** subnet if you don't need it.

3. Configure network environment. On the following form you can configure parameters of your network environment:

**TEMPLATE**



101-sql-managed-instance-azure-environment  
2 resources

[Edit template](#) [Edit parameters](#) [Learn more](#)

---

**BASICS**

|                  |  |
|------------------|--|
| * Subscription   | <input type="text" value="Pay-As-You-Go"/>   |
| * Resource group | <input checked="" type="radio"/> Create new <input type="radio"/> Use existing<br><input type="text" value="Create a resource group"/> |
| * Location       | <input type="text" value="West Central US"/>   |

---

**SETTINGS**

|  |   |
|--|---|
| Virtual Network Name            | <input type="text" value="MyNewVNet"/>                    |
| Virtual Network Address Prefix  | <input type="text" value="10.0.0.0/16"/>                  |
| Default Subnet Name             | <input type="text" value="Default"/>                      |
| Default Subnet Prefix           | <input type="text" value="10.0.0.0/24"/>                  |
| Managed Instance Subnet Name    | <input type="text" value="ManagedInstances"/>             |
| Managed Instance Subnet Prefix  | <input type="text" value="10.0.1.0/24"/>                  |
| Route Table To Azure Service  | <input type="text" value="RouteToAzureSqlMiMngSvc"/>      |
| Location                      | <input type="text" value="[\$resourceGroup().location]"/> |

Pin to dashboard

[Purchase](#)

You might change the names of VNet and subnets and adjust IP ranges associated to your networking resources. Once you press "Purchase" button, this form will create and configure your environment. If you don't need two subnets you can delete the default one.

## Modify an existing virtual network for Managed Instances

The questions and answers in this section show you how to add a Managed Instance to existing virtual network.

### Are you using Classic or Resource Manager deployment model for the existing virtual network?

You can only create a Managed Instance in Resource Manager virtual networks.

### Would you like to create new subnet for SQL Managed instance or use existing one?

If you would like to create new one:

- Calculate subnet size by following the guidelines in the [Determine the size of subnet for Managed Instances](#) section.
- Follow the steps in [Add, change, or delete a virtual network subnet](#).
- Create a route table that contains single entry, **0.0.0.0/0**, as the next hop Internet and associate it with the subnet for the Managed Instance.

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
```

Subnet preparation is done in three simple steps:

- Validate - Selected virtual network and subnet are validated for Managed Instance networking requirements
- Confirm - User is shown a set of changes that need to be made to prepare subnet for Managed Instance deployment and asked for consent
- Prepare - Virtual network and subnet are configured properly

#### **Do you have custom DNS server configured?**

If yes, see [Configuring a Custom DNS](#).

## Next steps

- For an overview, see [What is a Managed Instance](#)
- For a tutorial showing how to create a VNet, create a Managed Instance, and restore a database from a database backup, see [Creating an Azure SQL Database Managed Instance](#).
- For DNS issues, see [Configuring a Custom DNS](#)

# Configuring a Custom DNS for Azure SQL Database Managed Instance

9/26/2018 • 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 (i.e. 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. Since Managed Instance uses the same DNS for its inner workings, the virtual network DNS configuration needs to be compatible with Managed Instance.

To make a custom DNS configuration compatible with the Managed Instance, you need to:

- Configure custom DNS server so it is able to resolve public domain names
- Put Azure Recursive Resolver DNS IP address 168.63.129.16 at the end of the virtual network DNS list

## Setting up custom DNS servers configuration

1. In the Azure portal, find custom DNS option for your VNet.

The screenshot shows the Azure portal interface for managing a virtual network named 'VNET\_MIPlayground'. On the left, a list of virtual networks is displayed, with 'VNET\_MIPlayground' selected and highlighted by a red box. On the right, the 'DNS servers' configuration page is shown. Under the 'DNS servers' section, there are two options: 'Default (Azure-provided)' (selected) and 'Custom'. Below this, a link to 'DNS servers' is also highlighted with a red box. The page includes standard navigation and save/discard buttons.

2. Switch to Custom and enter your custom DNS server IP address as well as Azure's recursive resolvers IP address 168.63.129.16.

The screenshot shows the Azure portal interface for managing DNS servers in a virtual network. On the left, a sidebar lists various settings like Overview, Activity log, and DNS servers, with 'DNS servers' highlighted by a red box. The main pane shows the 'Custom' DNS server configuration, with two entries listed: '10.0.1.5' and '168.63.129.16'. An 'Add DNS server' input field is also present.

#### IMPORTANT

Not setting Azure's recursive resolver in the DNS list can cause the Managed Instance to enter a faulty state when the custom DNS servers are unavailable for some reason. Recovering from that state may require you to create new instance in a VNet with the compliant networking policies, create instance level data, and restore your databases. Setting the Azure's recursive resolver as the last entry in the DNS list ensures, even when all custom DNS servers fail, public names can still be resolved. See [VNet Configuration](#).

## 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

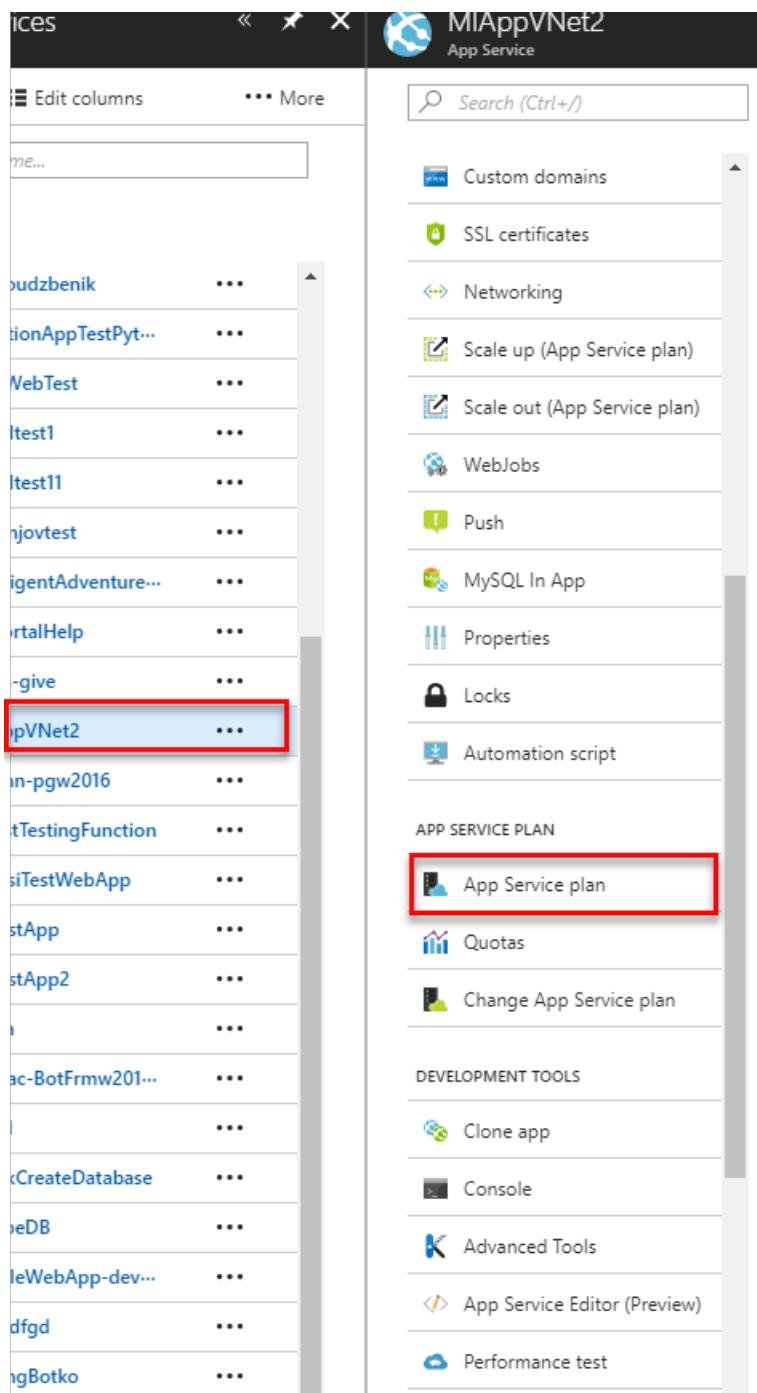
9/25/2018 • 2 minutes to read • [Edit Online](#)

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**.

Home > App Services > MIAppVNet2 > MITestApp - Networking

## MITestApp - Networking

App Service plan

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

**SETTINGS**

- Apps
- File system storage
- Networking** (highlighted with a red box)
- Scale up (App Service plan)
- Scale out (App Service plan)
- Properties
- Locks
- Automation script

**VNET Integration**  
Loading...

Securely access resources available in or through your Azure VNET

[Click here to manage!](#) (highlighted with a red box)

**Hybrid connections**

Securely access applications in private networks

[Learn More](#)

Configure your hybrid connection endpoints

3. Select your **VNet** and click **Sync Network**.

Home > App Services > MIAppVNet2 > MITestApp - Networking > Virtual Network Integration > Virtual Network Integration

## Virtual Network Integration

MITestApp

| VNET NAME              | GATEWAY STATUS |
|------------------------|----------------|
| VNET_MIPlayground_Peer | Online         |
| vNet1-eastus           | Online         |

**Virtual Network Integrati... MITestApp**

**Save** **Discard** **Sync Network** (highlighted with a red box)

| VNET NAME              | LOCATION | CERTIFICATE STATUS   | GATEWAY STATUS |
|------------------------|----------|----------------------|----------------|
| VNET_MIPlayground_Peer | East US  | Certificates in sync | Online         |

**APPS USING VIRTUAL NETWORK**

- MITestApp2

**VNET ADDRESS SPACE**

| START ADDRESS | END ADDRESS |
|---------------|-------------|
| 10.3.0.0      | 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 configuration](#).

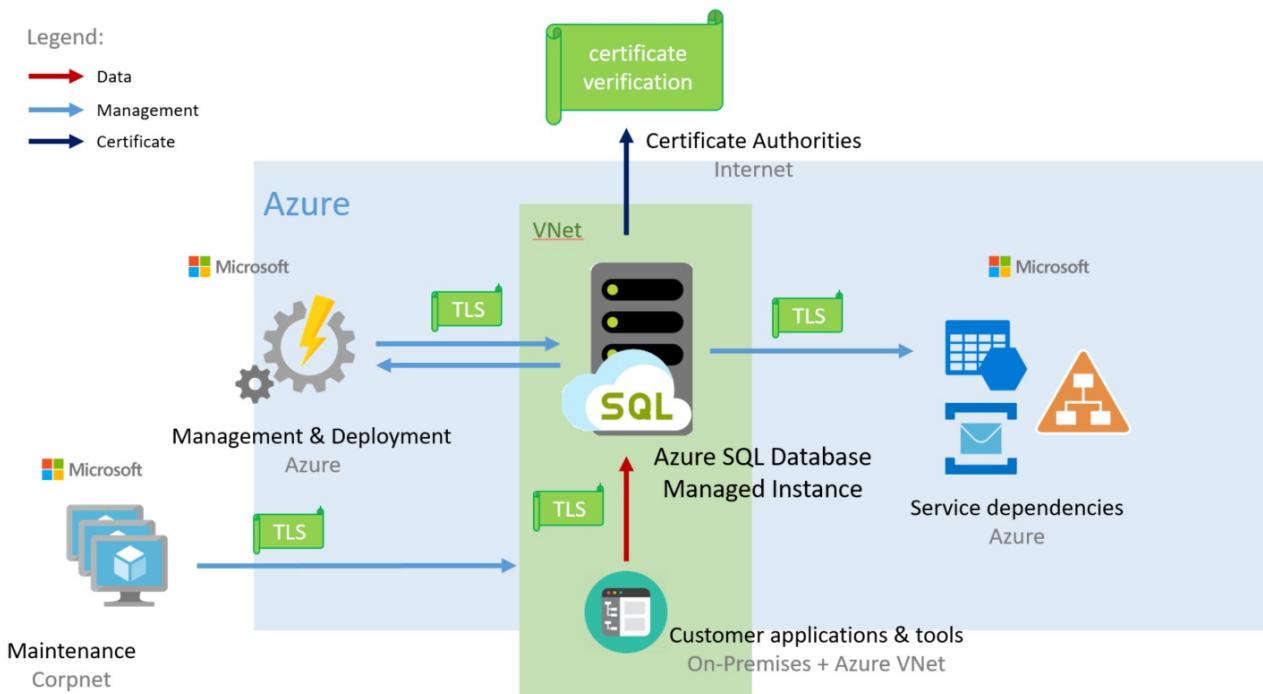
# Azure SQL Database Managed Instance Connectivity Architecture

9/25/2018 • 3 minutes to read • [Edit Online](#)

This article provides the Azure SQL Database Managed Instance communication overview and explains connectivity architecture as well as how the different components function to direct traffic to the Managed Instance.

## Communication overview

The following diagram shows entities that connect to Managed Instance as well as resources that Managed Instance has to reach out in order to function properly.



Communication that is depicted on the bottom of the diagram represents customer applications and tools connecting to Managed Instance as data source.

As Managed Instance is platform-as-a-services (PaaS) offering, Microsoft manages this service using automated agents (management, deployment, and maintenance) based on telemetry data streams. As Managed Instance management is solely Microsoft responsibility, customers are not able to access Managed Instance virtual cluster machines through RDP.

Some SQL Server operations initiated by the end users or applications may require Managed Instance to interact with the platform. One case is the creation of a Managed Instance database - a resource that is exposed through the portal, PowerShell, and Azure CLI.

Managed Instance depends on other Azure Services for its proper functioning (such as Azure Storage for backups, Azure Service Bus for telemetry, Azure AD for authentication, Azure Key Vault for TDE, and so forth) and initiates connections to them accordingly.

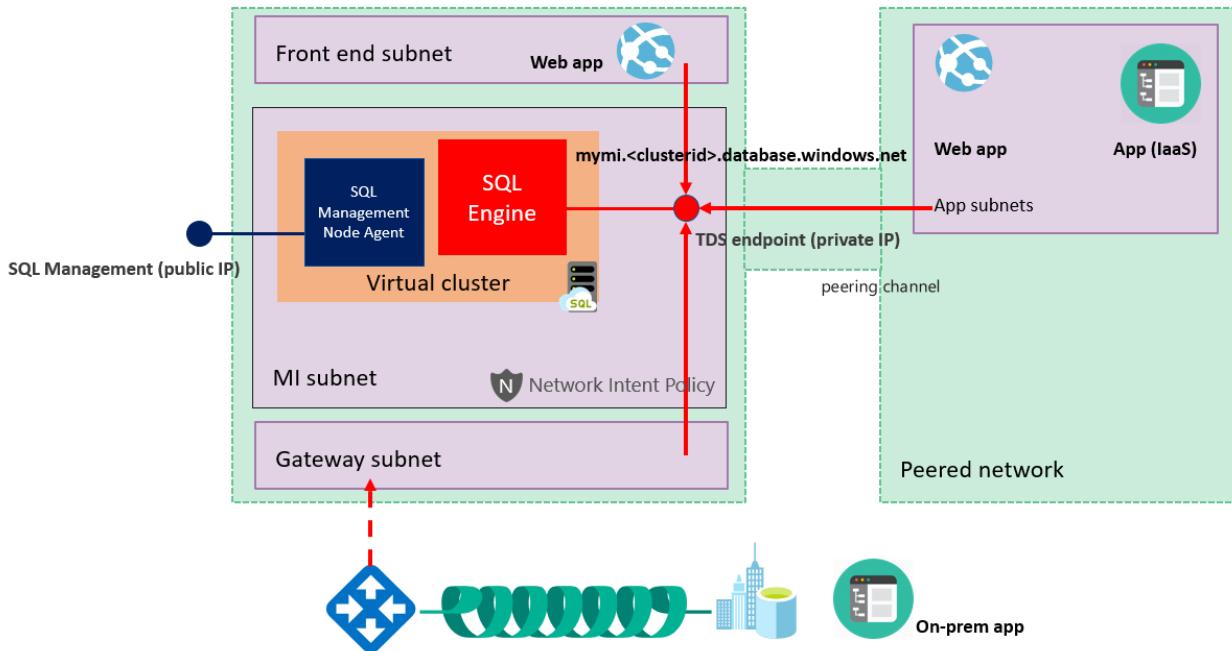
All communications, stated above, are encrypted and signed using certificates. To make sure that communicating parties are trusted, Managed Instance constantly verifies these certificates by contacting Certificate Authority. If the certificates are revoked or Managed Instance could not verify them, it closes the connections to protect the data.

# High-level connectivity architecture

At a high level, Managed Instance is a set of service components, hosted on a dedicated set of isolated virtual machines that run inside the customer virtual network subnet and form a virtual cluster.

Multiple Managed Instances could be hosted in single virtual cluster. The cluster is automatically expanded or contracted if needed when the customer changes the number of provisioned instances in the subnet.

Customer applications could connect to Managed Instance, query and update databases only if they run inside the virtual network or peered virtual network or VPN / Express Route connected network using endpoint with private IP address.



Microsoft management and deployment services run outside of the virtual network so connection between Managed Instance and Microsoft services goes over the endpoints with public IP addresses. When Managed Instance creates outbound connection, on receiving end it looks like it's coming from this public IP due to Network Address Translation (NAT).

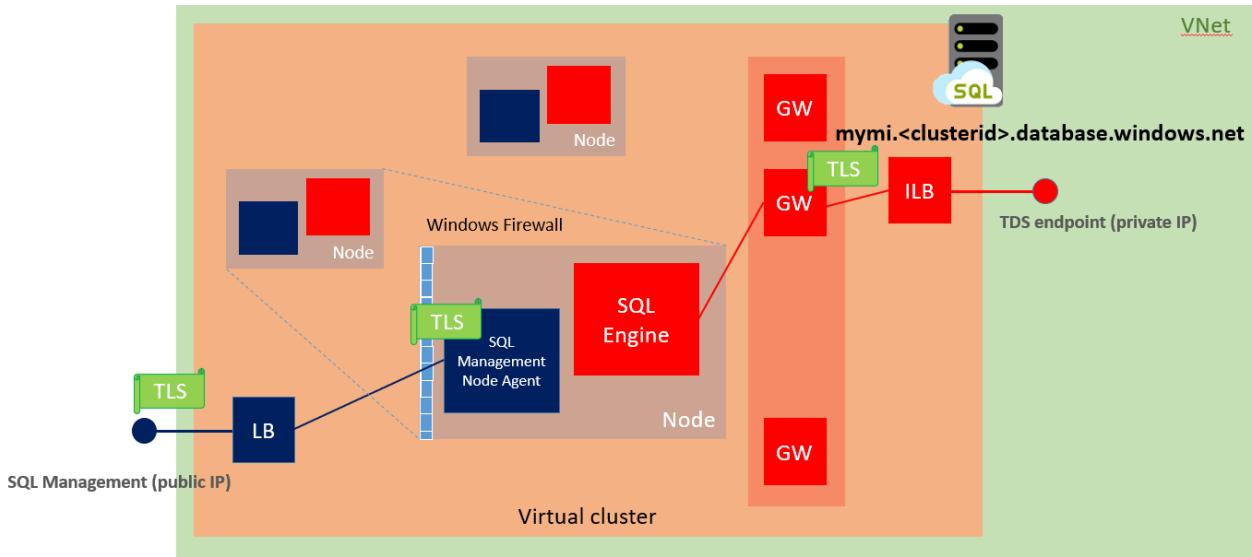
Management traffic flows through the customer virtual network. That means that elements of virtual network infrastructure affect and could potentially harm management traffic causing instance to enter faulty state and become unavailable.

## IMPORTANT

To improve customer experience and service availability, Microsoft applies Network Intent Policy on Azure virtual network infrastructure elements that could affect Managed Instance functioning. This is a platform mechanism to communicate transparently networking requirements to end users, with main goal to prevent network misconfiguration and ensure normal Managed Instance operations. Upon Managed Instance deletion Network Intent Policy is removed as well.

# Virtual cluster connectivity architecture

Let's take a deeper dive in Managed Instance connectivity architecture. The following diagram shows the conceptual layout of the virtual cluster.



Clients connect to Managed Instance using the host name that has a form <mi\_name>.database.windows.net. This host name resolves to private IP address although it is registered in public DNS zone and is publicly resolvable.

This private IP address belongs to the Managed Instance Internal Load Balancer (ILB) that directs traffic to the Managed Instance Gateway (GW). As multiple Managed Instances could potentially run inside the same cluster, GW uses Managed Instance host name to redirect traffic to the correct SQL Engine service.

Management and deployment services connect to Managed Instance using public endpoint that maps to external load balancer. Traffic is routed to the nodes only if received on predefined a set of ports that are used exclusively by Managed Instance management components. All communication between management components and management plane is mutually certificate authenticated.

## Next steps

- For an overview, see [What is a Managed Instance](#)
- For more information about VNet configuration, see [Managed Instance VNet Configuration](#).
- For a quickstart see how to create Managed Instance:
  - From the [Azure portal](#)
  - Using [PowerShell](#)
  - using [Azure Resource Manager template](#)
  - using [Azure Resource Manager template \(jumpbox with SSMS included\)](#)

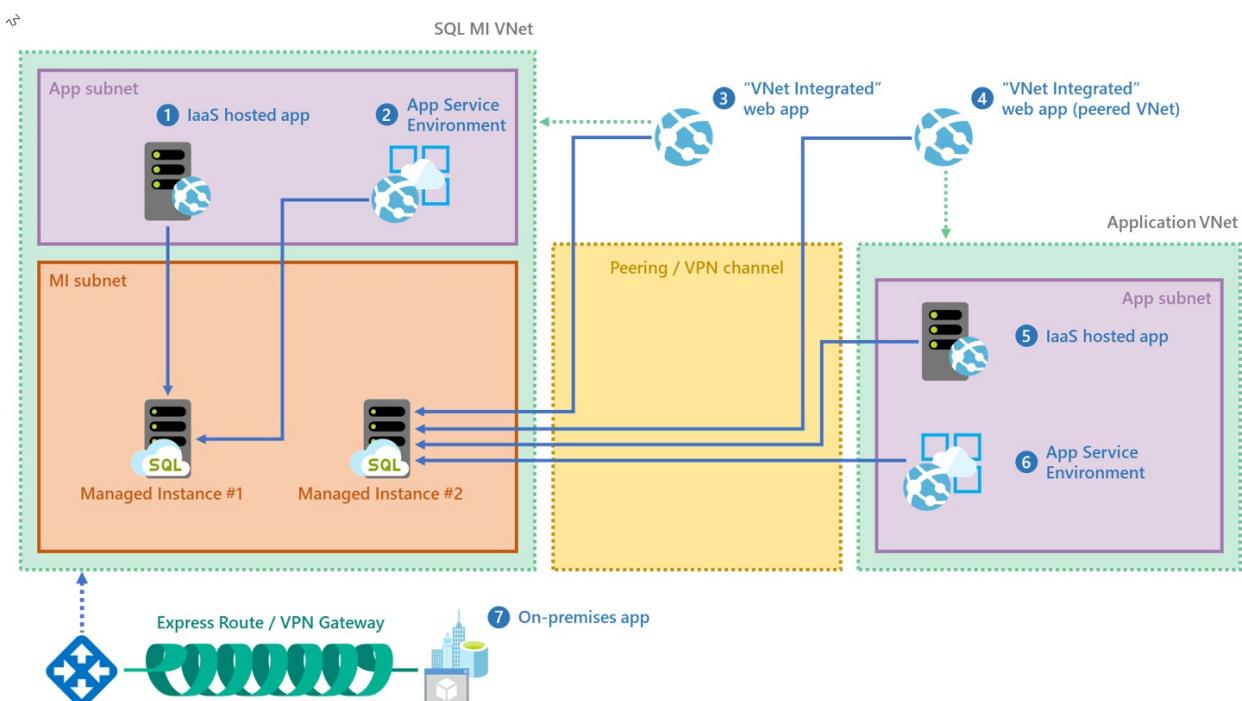
# Connect your application to Azure SQL Database Managed Instance

10/8/2018 • 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](#).

## 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

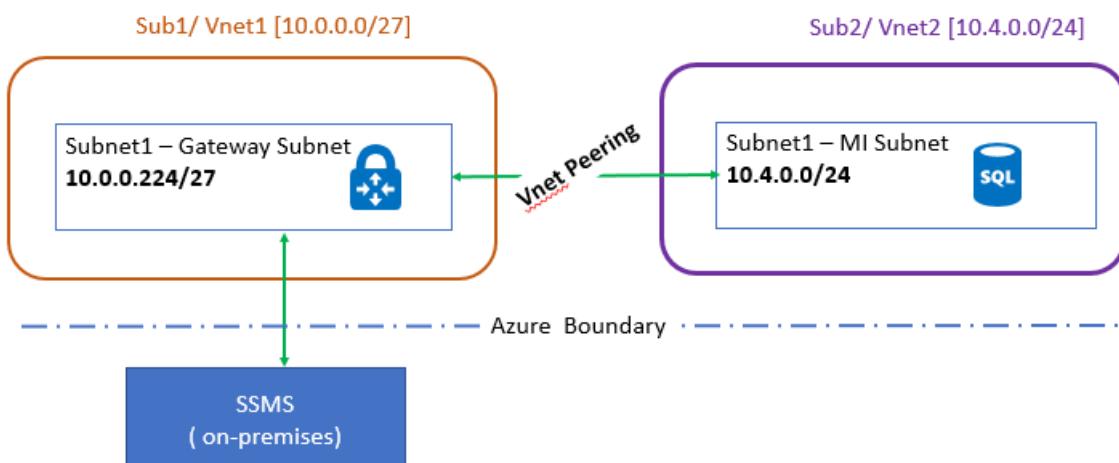
If you've established on-premises to Azure connection successfully and you can't establish connection to Managed Instance, check if your firewall has open outbound connection on SQL port 1433 as well as 11000-12000 range of ports for redirection.

## 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

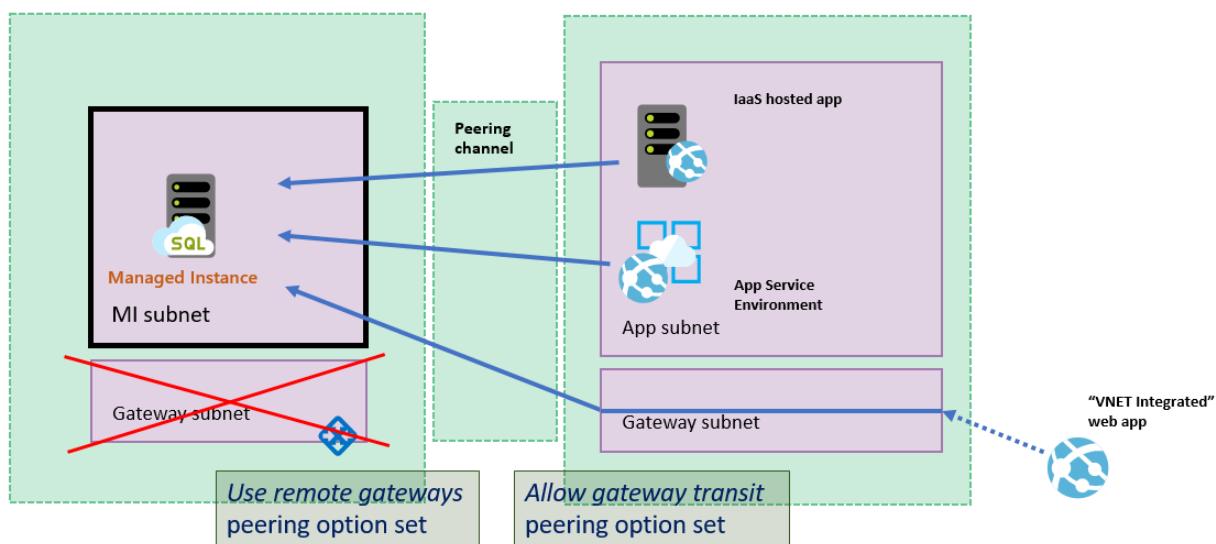
Managed Instance can be accessed only through a private IP address so in order to access it from Azure App Service you first need to make a connection between the application and the Managed Instance VNet. See [Integrate your app with an Azure Virtual Network](#).

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:



## 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 11000-12000 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)

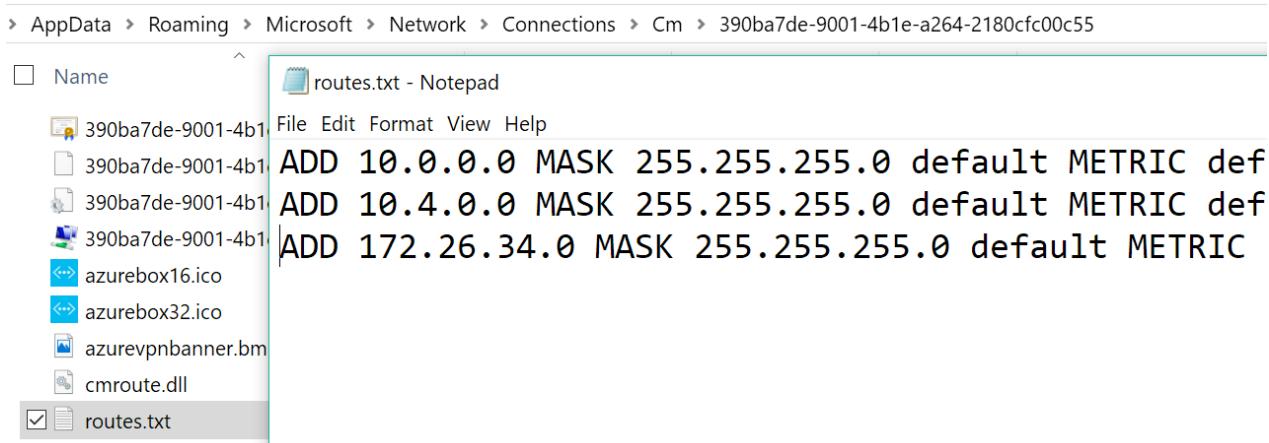
IKEv2 VPN

## 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 39 .....Intel(R) Ethernet Connection (3) I218-LM
57.....rndatavnet
18...94 65 9c 7d e5 ce .....Intel(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           | 17.8.1 or <a href="#">higher</a> |

## 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 SQL Database Managed Instance T-SQL differences from SQL Server

10/30/2018 • 15 minutes to read • [Edit Online](#)

Azure SQL Database Managed Instance provides high compatibility with on-premises SQL Server Database Engine. Most of the SQL Server Database Engine features are supported in Managed Instance. Since there are still some differences in syntax and behavior, this article summarizes and explains these differences.

- [T-SQL differences and unsupported features](#)
- [Features that have different behavior in Managed Instance](#)
- [Temporary limitations and known issues](#)

## T-SQL differences from SQL Server

This section summarizes key differences in T-SQL syntax and behavior between Managed Instance and on-premises SQL Server Database Engine, as well as unsupported features.

### Always-On availability

[High availability](#) is built into Managed Instance and cannot be controlled by users. The following statements are not supported:

- [CREATE ENDPOINT ... FOR DATABASE\\_MIRRORING](#)
- [CREATE AVAILABILITY GROUP](#)
- [ALTER AVAILABILITY GROUP](#)
- [DROP AVAILABILITY GROUP](#)
- [SET HADR clause of the ALTER DATABASE statement](#)

### Auditing

The key differences between SQL Audit in Managed Instance, Azure SQL Database, and SQL Server on-premises are:

- In Managed Instance, SQL Audit works at the server level and stores `.xel` files on Azure blob storage account.
- In Azure SQL Database, SQL Audit works at the database level.
- In SQL Server on-premises / virtual machine, SQL 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 `.xel` files will be placed
- The syntax `TO FILE` is not supported because Managed Instance cannot access Windows file shares.

For more information, see:

- [CREATE SERVER AUDIT](#)
- [ALTER SERVER AUDIT](#)

- [Auditing](#)

## Backup

Managed Instance has automatic backups, and enables users to create full database `COPY_ONLY` backups.

Differential, log, and file snapshot backups are not supported.

- Managed Instance can back up a database only to an Azure Blob Storage account:
  - Only `BACKUP TO URL` is supported
  - `FILE`, `TAPE`, and backup devices are not supported
- Most of the general `WITH` options are supported
  - `COPY_ONLY` is mandatory
  - `FILE_SNAPSHOT` not supported
  - Tape options: `REWIND`, `NOREWIND`, `UNLOAD`, and `NOUNLOAD` are not supported
  - Log-specific options: `NORECOVERY`, `STANDBY`, and `NO_TRUNCATE` are not supported

Limitations:

- Managed Instance can back up a database to a backup with up to 32 stripes, which is enough for the databases up to 4 TB if backup compression is used.
- Max backup stripe size is 195 GB (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 on-premises, backup to `DISK` instead of backup to `URL`, upload backup file to blob, then restore. Restore supports bigger files because a different blob type is used.

For information about backups using T-SQL, see [BACKUP](#).

## Buffer pool extension

- [Buffer pool extension](#) is not supported.
- `ALTER SERVER CONFIGURATION SET BUFFER POOL EXTENSION` is not supported. See [ALTER SERVER CONFIGURATION](#).

## Bulk insert / openrowset

Managed Instance cannot access file shares and Windows folders, so the files must be imported from Azure blob storage:

- `DATASOURCE` is required in `BULK INSERT` command while importing files from Azure blob storage. See [BULK INSERT](#).
- `DATASOURCE` is required in `OPENROWSET` function when you read a content of a file from Azure blob storage. See [OPENROWSET](#).

## Certificates

Managed Instance cannot access file shares and Windows folders, so the following constraints apply:

- `CREATE FROM / BACKUP TO` file is not supported for certificates
- `CREATE / BACKUP` certificate from `FILE / ASSEMBLY` is not supported. Private key files cannot be used.

See [CREATE CERTIFICATE](#) and [BACKUP CERTIFICATE](#).

**Workaround:** script certificate/private key, store as .sql file and create from binary:

```
CREATE CERTIFICATE
    FROM BINARY = asn_encoded_certificate
    WITH PRIVATE KEY (<private_key_options>)
```

## CLR

Managed Instance cannot 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` is not supported. See [CREATE ASSEMBLY FROM FILE](#).
- `ALTER ASSEMBLY` cannot reference files. See [ALTER ASSEMBLY](#).

## Compatibility levels

- Supported compatibility levels are: 100, 110, 120, 130, 140
- Compatibility levels below 100 are not supported.
- The default compatibility level for new databases is 140. For restored databases, compatibility level will remain unchanged if it was 100 and above.

See [ALTER DATABASE Compatibility Level](#).

## Credential

Only Azure Key Vault and `SHARED ACCESS SIGNATURE` identities are supported. Windows users are not supported.

See [CREATE CREDENTIAL](#) and [ALTER CREDENTIAL](#).

## Cryptographic providers

Managed Instance cannot access files so cryptographic providers cannot be created:

- `CREATE CRYPTOGRAPHIC PROVIDER` is not supported. See [CREATE CRYPTOGRAPHIC PROVIDER](#).
- `ALTER CRYPTOGRAPHIC PROVIDER` is not supported. See [ALTER CRYPTOGRAPHIC PROVIDER](#).

## Collation

Server collation is `SQL_Latin1_General_CI_AS` and cannot be changed. See [Collations](#).

## Database options

- Multiple log files are not supported.
- In-memory objects are not supported in the General Purpose service tier.
- There is a limit of 280 files per instance implying max 280 files per database. Both data and log files are counted toward this limit.
- Database cannot contain filegroups containing filestream data. Restore will fail if .bak contains `FILESTREAM` data.
- Every file is placed in Azure Premium storage. IO and throughput per file depend on the size of each individual file, in the same way as they do for Azure Premium Storage disks. See [Azure Premium disk performance](#)

## CREATE DATABASE statement

The following are `CREATE DATABASE` limitations:

- Files and filegroups cannot be defined.
- `CONTAINMENT` option is not supported.
- `WITH` options are not supported.

### TIP

As workaround, use `ALTER DATABASE` after `CREATE DATABASE` to set database options to add files or to set containment.

- `FOR ATTACH` option is not supported
- `AS SNAPSHOT OF` option is not supported

For more information, see [CREATE DATABASE](#).

### **ALTER DATABASE statement**

Some file properties cannot be set or changed:

- File path cannot be specified in `ALTER DATABASE ADD FILE (FILENAME='path')` T-SQL statement. Remove `FILENAME` from the script because Managed Instance automatically places the files.
- File name cannot be changed using `ALTER DATABASE` statement.

The following options are set by default and cannot be changed:

- `MULTI_USER`
- `ENABLE_BROKER ON`
- `AUTO_CLOSE OFF`

The following options cannot 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`

Modify name is not supported.

For more information, see [ALTER DATABASE](#).

### **Database mirroring**

Database mirroring is not supported.

- `ALTER DATABASE SET PARTNER` and `SET WITNESS` options are not supported.
- `CREATE ENDPOINT ... FOR DATABASE_MIRRORING` is not supported.

For more information, see [ALTER DATABASE SET PARTNER](#) and [SET WITNESS](#) and [CREATE ENDPOINT ... FOR DATABASE\\_MIRRORING](#).

## DBCC

Undocumented DBCC statements that are enabled in SQL Server are not supported in Managed Instance.

- `Trace Flags` are not supported. See [Trace Flags](#).
- `DBCC TRACEOFF` is not supported. See [DBCC TRACEOFF](#).
- `DBCC TRACEON` is not supported. See [DBCC TRACEON](#).

## Distributed transactions

Neither MSDTC nor [Elastic Transactions](#) are currently supported in Managed Instance.

## Extended Events

Some Windows-specific targets for XEvents are not supported:

- `etw_classic_sync target` is not supported. Store `.xel` files on Azure blob storage. See [etw\\_classic\\_sync target](#).
- `event_file target` is not supported. Store `.xel` files on Azure blob storage. See [event\\_file target](#).

## External libraries

In-database R and Python external libraries are not yet supported. See [SQL Server Machine Learning Services](#).

## Filestream and Filetable

- filestream data is not supported.
- Database cannot contain filegroups with `FILESTREAM` data
- `FILETABLE` is not supported
- Tables cannot have `FILESTREAM` types
- The following functions are not supported:
  - `GetPathLocator()`
  - `GET_FILESTREAM_TRANSACTION_CONTEXT()`
  - `PathName()`
  - `GetFileNamespacePat()`
  - `FileTableRootPath()`

For more information, see [FILESTREAM](#) and [FileTables](#).

## Full-text Semantic Search

[Semantic Search](#) is not supported.

## Linked servers

Linked servers in Managed Instance support a limited number of targets:

- Supported targets: SQL Server and SQL Database
- Not supported targets: files, Analysis Services, and other RDBMS.

## Operations

- Cross-instance write transactions are not supported.
- `sp_dropserver` is supported for dropping a linked server. See [sp\\_dropserver](#).
- `OPENROWSET` function can be used to execute queries only on SQL Server instances (either managed, on-

premises, or in Virtual Machines). See [OPENROWSET](#).

- `OPENDATASOURCE` function can be used to execute queries only on SQL Server instances (either managed, on-premises, or in virtual machines). Only `SQLNCLI`, `SQLNCLI11`, and `SQLOLEDB` values are supported as provider. For example: `SELECT * FROM OPENDATASOURCE('SQLNCLI', '...').AdventureWorks2012.HumanResources.Employee`. See [OPENDATASOURCE](#).

## Logins / users

- SQL logins created `FROM CERTIFICATE`, `FROM ASYMMETRIC KEY`, and `FROM SID` are supported. See [CREATE LOGIN](#).
- Windows logins created with `CREATE LOGIN ... FROM WINDOWS` syntax are not supported.
- Azure Active Directory (Azure AD) user who created the instance has [unrestricted admin privileges](#).
- Non-administrator Azure Active Directory (Azure AD) database-level users can be created using `CREATE USER ... FROM EXTERNAL PROVIDER` syntax. See [CREATE USER ... FROM EXTERNAL PROVIDER](#)

## Polybase

External tables referencing the files in HDFS or Azure blob storage are not supported. For information about Polybase, see [Polybase](#).

## Replication

Replication is available for public preview on Managed Instance. For information about Replication, see [SQL Server Replication](#).

## 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 only supported option.
  - `FROM DISK` / `TAPE` /backup device is not supported.
  - Backup sets are not supported.
- `WITH` options are not supported (No `DIFFERENTIAL`, `STATS`, etc.)
- `ASYNC RESTORE` - Restore continues even if client connection breaks. If your connection is dropped, you can check `sys.dm_operation_status` view for the status of a restore operation (as well as for CREATE and DROP database). See [sys.dm\\_operation\\_status](#).

The following database options are set/overridden and cannot be changed later:

- `NEW_BROKER` (if broker is not enabled in .bak file)
- `ENABLE_BROKER` (if broker is not enabled in .bak file)
- `AUTO_CLOSE=OFF` (if a database in .bak file has `AUTO_CLOSE=ON`)
- `RECOVERY FULL` (if a database in .bak file has `SIMPLE` or `BULK_LOGGED` recovery mode)
- Memory optimized filegroup is added and called XTP if it was not 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:

- `.BAK` files containing multiple backup sets cannot be restored.
- `.BAK` files containing multiple log files cannot be restored.
- Restore will fail if `.bak` contains `FILESTREAM` data.
- Backups containing databases that have active In-memory objects cannot currently be restored.
- Backups containing databases where at some point In-Memory objects existed cannot currently be restored.
- Backups containing databases in read-only mode cannot currently be restored. This limitation will be removed soon.

For information about Restore statements, see [RESTORE Statements](#).

## Service broker

Cross-instance service broker is not supported:

- `sys.routes` - Prerequisite: select address from `sys.routes`. Address must be LOCAL on every route. See [sys.routes](#).
- `CREATE ROUTE` - you cannot `CREATE ROUTE` with `ADDRESS` other than `LOCAL`. See [CREATE ROUTE](#).
- `ALTER ROUTE` cannot `ALTER ROUTE` with `ADDRESS` other than `LOCAL`. See [ALTER ROUTE](#).

## Service key and service master key

- `Master key backup` is not supported (managed by SQL Database service)
- `Master key restore` is not supported (managed by SQL Database service)
- `Service master key backup` is not supported (managed by SQL Database service)
- `Service master key restore` is not supported (managed by SQL Database service)

## Stored procedures, functions, triggers

- `NATIVE_COMPILATION` is currently not supported.
- The following `sp_configure` options are not supported:
  - `allow polybase export`
  - `allow updates`
  - `filestream_access_level`
  - `max text repl size`
  - `remote data archive`
  - `remote proc trans`
- `sp_execute_external_scripts` is not supported. See [sp\\_execute\\_external\\_scripts](#).
- `xp_cmdshell` is not supported. See [xp\\_cmdshell](#).
- `Extended stored procedures` are not supported, including `sp_addextendedproc` and `sp_dropextendedproc`. See [Extended stored procedures](#)
- `sp_attach_db`, `sp_attach_single_file_db`, and `sp_detach_db` are not supported. See [sp\\_attach\\_db](#), [sp\\_attach\\_single\\_file\\_db](#), and [sp\\_detach\\_db](#).
- `sp_renamedb` is not supported. See [sp\\_renamedb](#).

## SQL Server Agent

- SQL Agent settings are read only. Procedure `sp_set_agent_properties` is not 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 are not currently supported, including:
  - Merge replication job step is not supported.
  - Queue Reader is not supported.
  - Command shell is not yet supported
- Managed Instance cannot access external resources (for example, network shares via robocopy).
- PowerShell is not yet supported.
- Analysis Services are not supported
- Notifications are partially supported
- Email notification is supported, requires configuring a Database Mail profile. There can be only one database mail profile and it must be called `AzureManagedInstance_dbmail_profile` in public preview (temporary limitation).
  - Pager is not supported.
  - NetSend is not supported.
  - Alerts are not yet not supported.
  - Proxies are not supported.
- Eventlog is not supported.

The following features are currently not supported but will be enabled in future:

- Proxies
- Scheduling jobs on idle CPU
- Enabling/disabling Agent
- Alerts

For information about SQL Server Agent, see [SQL Server Agent](#).

## Tables

The following are not supported:

- `FILESTREAM`
- `FILETABLE`
- `EXTERNAL TABLE`
- `MEMORY_OPTIMIZED`

For information about creating and altering tables, see [CREATE TABLE](#) and [ALTER TABLE](#).

## Behavior changes

The following variables, functions, and views return different results:

- `SERVERPROPERTY('EngineEdition')` returns value 8. This property uniquely identifies Managed Instance. See [SERVERPROPERTY](#).
- `SERVERPROPERTY('InstanceName')` returns NULL, because the concept of instance as it exists for SQL Server does not apply to Managed Instance. See [SERVERPROPERTY\('InstanceName'\)](#).
- `@@SERVERNAME` returns full DNS 'connectable' name, for example, my-managed-instance.wcus17662feb9ce98.database.windows.net. See [@@SERVERNAME](#).
- `SYS.SERVERS` - returns full DNS 'connectable' name, such as `myinstance.domain.database.windows.net` for properties 'name' and 'data\_source'. See [SYS.SERVERS](#).
- `@@SERVICENAME` returns NULL, because the concept of service as it exists for SQL Server does not apply to

Managed Instance. See [@@SERVICENAME](#).

- `SUSER_ID` is supported. Returns NULL if AAD login is not in sys.syslogins. See [SUSER\\_ID](#).
- `SUSER_SID` is not supported. Returns wrong data (temporary known issue). See [SUSER\\_SID](#).
- `GETDATE()` and other built-in date/time functions always returns time in UTC time zone. See [GETDATE](#).

## Known issues and limitations

### TEMPDB size

`tempdb` is split into 12 files each with max size 14 GB per file. This maximum size per file cannot be changed and new files cannot be added to `tempdb`. This limitation will be removed soon. Some queries might return an error if they need more than 168 GB in `tempdb`.

### Exceeding storage space with small database files

Each Managed Instance has up to 35 TB storage reserved for Azure Premium Disk space, and 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 disk is not charged, but the total sum of Azure Premium Disk sizes cannot exceed 35 TB. In some cases, a Managed Instance that does not need 8 TB in total might exceed the 35 TB Azure limit on storage size, due to internal fragmentation.

For example, a Managed Instance could have one file 1.2 TB in size that is placed on a 4 TB disk, and 248 files each 1 GB in size that are placed on separate 128 GB disks. In this example:

- the total 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 illustrates that under certain circumstance, due to a very specific distribution of files, a Managed Instance might reach the 35TB reserved for attached Azure Premium Disk when you might not expect it to.

In this example existing databases will continue to work and can grow without any problem as long as new files are not added. However new databases could not be created or restored because there is not enough space for new disk drives, even if the total size of all databases does not reach the instance size limit. The error that is returned in that case is not clear.

### Incorrect configuration of SAS key during database restore

`RESTORE DATABASE` that reads .bak file might be constantly retrying to read .bak file and return error after long period of time if Shared Access Signature in `CREDENTIAL` is incorrect. Execute `RESTORE HEADERONLY` before restoring a database to be sure that SAS key is correct. Make sure that you remove the leading `?` from the SAS key that is generated using Azure portal.

### Tooling

SQL Server Management Studio and SQL Server Data Tools might have some issues while accessing Managed Instance. All tooling issues will be addressed before General Availability.

### Incorrect database names in some views, logs, and messages

Several system views, performance counters, error messages, XEvents, and error log entries display GUID database identifiers instead of the actual database names. Do not rely on these GUID identifiers because they would be replaced with actual database names in the future.

### Database mail profile

There can be only one database mail profile and it must be called `AzureManagedInstance_dbmail_profile`. This is a temporary limitation that will be removed soon.

### Error logs are not-persisted

Error logs that are available in managed instance are not persisted and their size is not included in the max

storage limit. Error logs might be automatically erased in case of failover.

### Error logs are verbose

Managed Instance places verbose information in error logs and many of them are not relevant. The amount of information in error logs will be decreased in the future.

**Workaround:** Use a custom procedure for reading error logs that filter-out some non-relevant entries. For details, see [Azure SQL DB Managed Instance – sp\\_readmierrorlog](#).

### Transaction Scope on two databases within the same instance is not supported

`TransactionScope` class in .Net does not work if two queries are sent to the 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 other database in connection context instead of using two connections.

### CLR modules and linked servers sometime cannot reference local IP address

CLR modules placed in Managed Instance and linked servers/distributed queries that are referencing current instance sometime cannot resolve the IP of the local instance. This is transient error.

**Workaround:** Use context connections in CLR module if possible.

## Next steps

- For details about Managed Instance, see [What is a Managed Instance?](#)
- For a features and comparison list, see [SQL common features](#).
- For a quickstart showing you how to create a new Managed Instance, see [Creating a Managed Instance](#).

# Replication with SQL Database Managed Instance

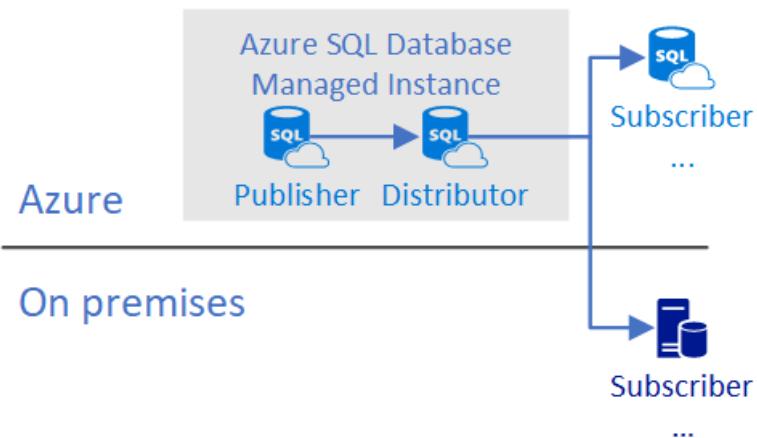
10/8/2018 • 3 minutes to read • [Edit Online](#)

Replication is available for public preview on [Azure SQL Database Managed Instance](#). A Managed Instance can host publisher, distributor, and subscriber databases.

## Common configurations

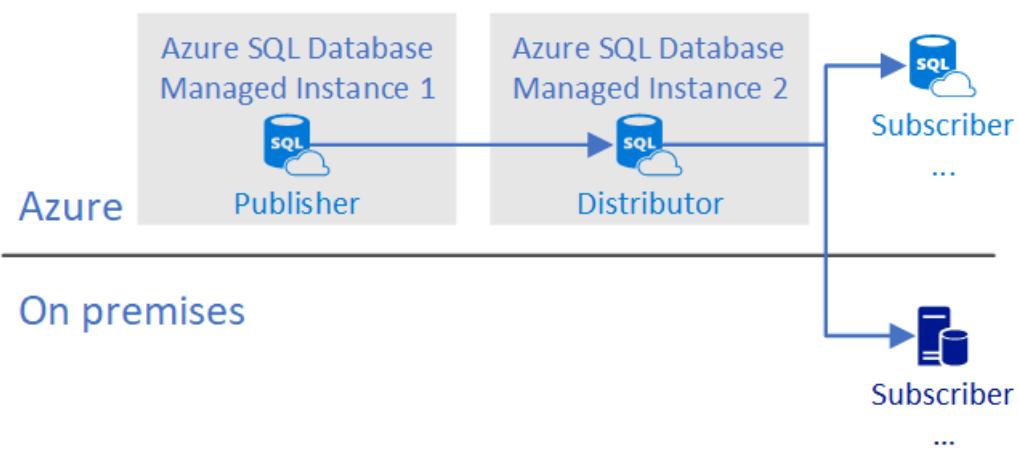
In general, the publisher and the distributor must both be either in the cloud or on-premises. The following configurations are supported:

- **Publisher with local distributor on managed instance**



Publisher and distributor databases are configured on a single managed instance.

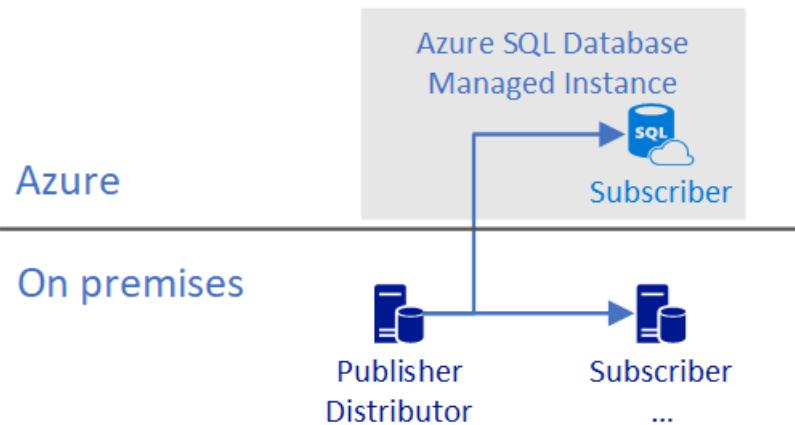
- **Publisher with remote distributor on managed instance**



Publisher and distributor are configured on two managed instances. In this configuration:

- Both managed instances are in the same vNet.
- Both managed instances are in the same location.

- **Publisher and distributor on-premises with subscriber on managed instance**



In this configuration, an Azure SQL database is a subscriber. This configuration supports migration from on-premises to Azure. In the subscriber role, SQL database does not require Managed Instance, however you can use a SQL Database Managed Instance as a step in migration from on-premises to the Azure. For more information about Azure SQL Database subscribers, see [Replication to SQL Database](#).

## Requirements

Publisher and distributor on Azure SQL Database requires:

- Azure SQL Database Managed Instance.

### NOTE

Azure SQL Databases that are not configured with Managed Instance can only be subscribers.

- All instances of SQL Server need to be on the same vNet.
- Connectivity uses SQL Authentication between replication participants.
- An Azure Storage Account share for the replication working directory.

## Features

Supports:

- Transactional and snapshot replication mix of on-premises and Azure SQL Database Managed Instance instances.
- Subscribers can be on-premises, single databases in Azure SQL Database, or pooled databases in Azure SQL Database elastic pools.
- One-way or bidirectional replication

## Configure publishing and distribution example

1. [Create an Azure SQL Database Managed Instance](#) in the portal.
2. [Create an Azure Storage Account](#) for the working directory.

Be sure to copy the storage keys. See [View and copy storage access keys](#).

3. Create a database for the publisher.

In the example scripts below, replace `<Publishing_DB>` with the name of this database.

4. Create a database user with SQL Authentication for the distributor. See, [Creating database users](#). Use a

secure password.

In the example scripts below, use `<SQL_USER>` and `<PASSWORD>` with this SQL Server Account database user and password.

5. [Connect to the SQL Database Managed Instance.](#)

6. Run the following query to add the distributor and the distribution database.

```
USE [master]
GO
EXEC sp_adddistributor @distributor = @@ServerName;
EXEC sp_adddistributiondb @database = N'distribution';
```

7. To configure a publisher to use a specified distribution database, update and run the following query.

Replace `<SQL_USER>` and `<PASSWORD>` with the SQL Server Account and password.

Replace `\\\<STORAGE_ACCOUNT>.file.core.windows.net\<SHARE>` with the value of your storage account.

Replace `<STORAGE_CONNECTION_STRING>` with the connection string from the **Access keys** tab of your Microsoft Azure storage account.

After you update the following query, run it.

```
USE [master]
EXEC sp_adddistpublisher @publisher = @@ServerName,
    @distribution_db = N'distribution',
    @security_mode = 0,
    @login = N'<SQL_USER>',
    @password = N'<PASSWORD>',
    @working_directory = N'\\<STORAGE_ACCOUNT>.file.core.windows.net\<SHARE>',
    @storage_connection_string = N'<STORAGE_CONNECTION_STRING>';
GO
```

8. Configure the publisher for replication.

In the following query, replace `<Publishing_DB>` with the name of your publisher database.

Replace `<Publication_Name>` with the name for your publication.

Replace `<SQL_USER>` and `<PASSWORD>` with the SQL Server Account and password.

After you update the query, run it to create the publication.

```

USE [<Publishing_DB>]
EXEC sp_replicationdboption @dbname = N'<Publishing_DB>',
    @optname = N'publish',
    @value = N'true';

EXEC sp_addpublication @publication = N'<Publication_Name>',
    @status = N'active';

EXEC sp_changelogreader_agent @publisher_security_mode = 0,
    @publisher_login = N'<SQL_USER>',
    @publisher_password = N'<PASSWORD>',
    @job_login = N'<SQL_USER>',
    @job_password = N'<PASSWORD>';

EXEC sp_addpublication_snapshot @publication = N'<Publication_Name>',
    @frequency_type = 1,
    @publisher_security_mode = 0,
    @publisher_login = N'<SQL_USER>',
    @publisher_password = N'<PASSWORD>',
    @job_login = N'<SQL_USER>',
    @job_password = N'<PASSWORD>'

```

## 9. Add the article, subscription, and push subscription agent.

To add these objects, update the following script.

- Replace `<Object_Name>` with the name of the publication object.
- Replace `<Object_Schema>` with the name of the source schema.
- Replace the other parameters in angle brackets `<>` to match the values in the previous scripts.

```

EXEC sp_addarticle @publication = N'<Publication_Name>',
    @type = N'logbased',
    @article = N'<Object_Name>',
    @source_object = N'<Object_Name>',
    @source_owner = N'<Object_Schema>'

EXEC sp_addsubscription @publication = N'<Publication_Name>',
    @subscriber = @@ServerName,
    @destination_db = N'<Subscribing_DB>',
    @subscription_type = N'Push'

EXEC sp_addpushsubscription_agent @publication = N'<Publication_Name>',
    @subscriber = @@ServerName,
    @subscriber_db = N'<Subscribing_DB>',
    @subscriber_security_mode = 0,
    @subscriber_login = N'<SQL_USER>',
    @subscriber_password = N'<PASSWORD>',
    @job_login = N'<SQL_USER>',
    @job_password = N'<PASSWORD>'

GO

```

## Limitations

The following features are not supported:

- Updateable subscriptions
- Active geo replication

## See Also

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

# Overview of business continuity with Azure SQL Database

10/24/2018 • 10 minutes to read • [Edit Online](#)

Azure SQL Database is an implementation of the latest stable SQL Server Database Engine configured and optimized for Azure cloud environment that provides [high availability](#) and resiliency to the errors that might affect your business process. **Business continuity** in Azure SQL Database refers to the mechanisms, policies, and procedures that enable a 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 business processes running. However, there are some disruptive events that cannot be handled by SQL Database 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.

These cases cannot be controlled by Azure SQL Database, so you would need to do use the business continuity features in SQL Database that enables you to recover your data and keep your applications running.

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 intrinsically application-specific and cannot as a rule be detected or mitigated automatically by the infrastructure.
- 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 upgrades or maintenance to an application or database may require rapid rollback to a prior database state.

SQL Database provides several business continuity features, including automated backups and optional database replication that can mitigate these scenarios. First, you need to understand how SQL Database [high availability architecture](#) provides 99.99% availability and resiliency to some disruptive events that might affect your business process. Then, you can learn about the additional mechanisms that you can use to recover from the disruptive events that cannot be handled by SQL Database high availability architecture, such as:

- [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 last 35 days.
- You can [restore a deleted database](#) to the point at which it was deleted if the **logical server has not been**

**deleted.**

- [Long-term backup retention](#) enables you to keep the backups up to 10 years.
- [Auto-failover group](#) allows the application to automatically recover in case of a data center scale outage.

Each has different characteristics for estimated recovery time (ERT) and potential data loss for recent transactions. Once you understand these options, you can choose among them - and, in most scenarios, use them together for different scenarios. 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 after the disruptive event. The time period of updates that you might afford to lose is known as recovery point objective (RPO).

The following table compares the ERT and RPO for each service tier for the three most common scenarios.

| CAPABILITY                              | BASIC                               | STANDARD                         | PREMIUM                          | GENERAL PURPOSE  | BUSINESS CRITICAL  |
|---|-------------------------------------|----------------------------------|----------------------------------|--|--|
| Point in Time Restore from backup       | Any restore point within seven days | Any restore point within 35 days | Any restore point within 35 days | Any restore point within configured period (up to 35 days) | Any restore point within configured period (up to 35 days) |
| Geo-restore from geo-replicated backups | ERT < 12 h<br>RPO < 1 h             | ERT < 12 h<br>RPO < 1 h          | ERT < 12 h<br>RPO < 1 h          | ERT < 12 h<br>RPO < 1 h                                    | ERT < 12 h<br>RPO < 1 h                                    |
| Auto-failover groups                    | RTO = 1 h<br>RPO < 5s               | RTO = 1 h<br>RPO < 5 s           | RTO = 1 h<br>RPO < 5 s           | RTO = 1 h<br>RPO < 5 s                                     | RTO = 1 h<br>RPO < 5 s                                     |

## Recover a database to the existing server

SQL Database automatically performs a combination of full database backups weekly, differential database backups generally taken every 12 hours, and transaction log backups every 5 - 10 minutes to protect your business from data loss. The backups are stored in RA-GRS storage for 35 days for all service tiers except Basic DTU service tiers where the backups are stored for 7 days. For more information, see [automatic database backups](#). You can restore an existing database from the automated backups to an earlier point in time as a new database on the same logical server using the Azure portal, PowerShell, or the REST API. For more information, see [Point-in-time restore](#).

If the maximum supported point-in-time restore (PITR) retention period 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](#).

You can use these automatic database backups to recover a database from various disruptive events, both within your data center and to another data center. The recovery time is usually less than 12 hours. It may take longer to recover a very large or active database. Using automatic database backups, the estimated time of recovery depends on several factors including the total number of databases recovering in the same region at the same time, the database size, the transaction log size, and network bandwidth. For more information about recovery time, see [database recovery time](#). When recovering to another data region, the potential data loss is limited to 1 hour with use of geo-redundant backups.

Use automated backups and [point-in-time restore](#) as your business continuity and recovery mechanism if your application:

- Is not considered mission critical.
- Doesn't have a binding SLA - a downtime of 24 hours or longer does not result in financial liability.
- Has a low rate of data change (low transactions per hour) and losing up to an hour of change is an acceptable data loss.
- Is cost sensitive.

If you need faster recovery, use [failover groups](#) (discussed next). If you need to be able to recover data from a period older than 35 days, use [Long-term retention](#).

## Recover a database to another region

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 a [auto-failover group](#) for your database or databases. You can customize the failover policy to use automatic or manual 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. See the table earlier in this article for details of the auto-failover RTO and RPO.

### **IMPORTANT**

To use active geo-replication and auto-failover groups, you must either be the subscription owner or have administrative permissions in SQL Server. You can configure and fail over using the Azure portal, PowerShell, or the REST API using Azure subscription permissions or using Transact-SQL with SQL Server permissions.

Use active 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.

When you take action, how long it takes you to recover, and how much data loss you incur depends upon how you decide to use these business continuity features in your application. Indeed, you may choose to use a combination of database backups and active geo-replication depending upon your application requirements. For a discussion of application 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 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 and auto-failover groups as your recovery mechanism, you can configure an automatic failover policy or use [manual 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 firewall rules are in place for users to connect (or use [database-level firewalls](#))
- 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

10/16/2018 • 7 minutes to read • [Edit Online](#)

Azure SQL Database is highly available database Platform as a Service that guarantees that your database is up and running 99.99% of time, without worrying about maintenance and downtimes. This is a fully managed SQL Server Database Engine process hosted in the Azure cloud that ensures that your SQL Server database is always upgraded/patched without affecting your workload. When an instance is patched or fails over, the downtime is generally not noticeable if you [employ retry logic](#) in your app. If the time to complete a failover is longer than 60 seconds, you should open a support case. Azure SQL Database can quickly recover even in the most critical circumstances ensuring that your data is always available.

Azure platform fully manages every Azure SQL Database and 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. SQL Server engineers have implemented the best-known practices, ensuring that all the maintenance operations are completed in less than 0.01% time of your database life. This architecture is designed to ensure that committed data is never lost and that maintenance operations are performed without affecting workload. There are no maintenance windows or downtimes that should require you to stop the workload while the database is upgraded or maintained. Built-in high availability in Azure SQL Database guarantees that database will never be single point of failure in your software architecture.

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 two high-availability architectural models that are used in Azure SQL Database (both of them ensuring 99.99% availability):

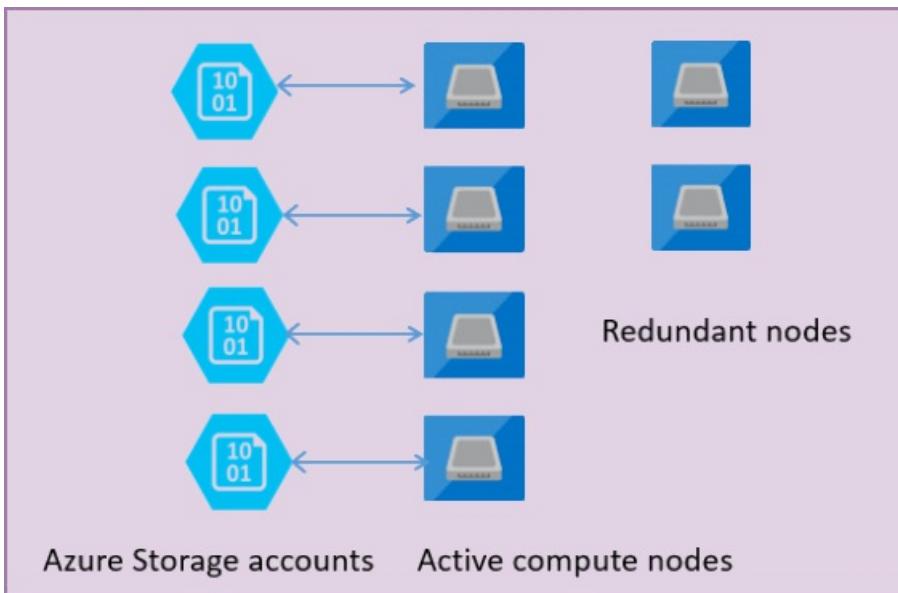
- Standard/general purpose service tier model that is based on a separation of compute and storage. This architectural model relies on high availability and reliability of storage tier, but it might have some potential performance degradation during maintenance activities.
- Premium/business critical service tier model that 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. Azure SQL Database runs on the latest stable version of SQL Server Database Engine and Windows OS, and most of the users would not notice that the upgrades are performed continuously.

## Basic, Standard, and General Purpose service tier availability

Standard availability refers to 99.99% SLA that is applied in Basic, Standard, and General Purpose service tiers. High availability in this architectural model is achieved by separation of compute and storage layers and the replication of data in the storage tier.

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



In the standard availability model there are two layers:

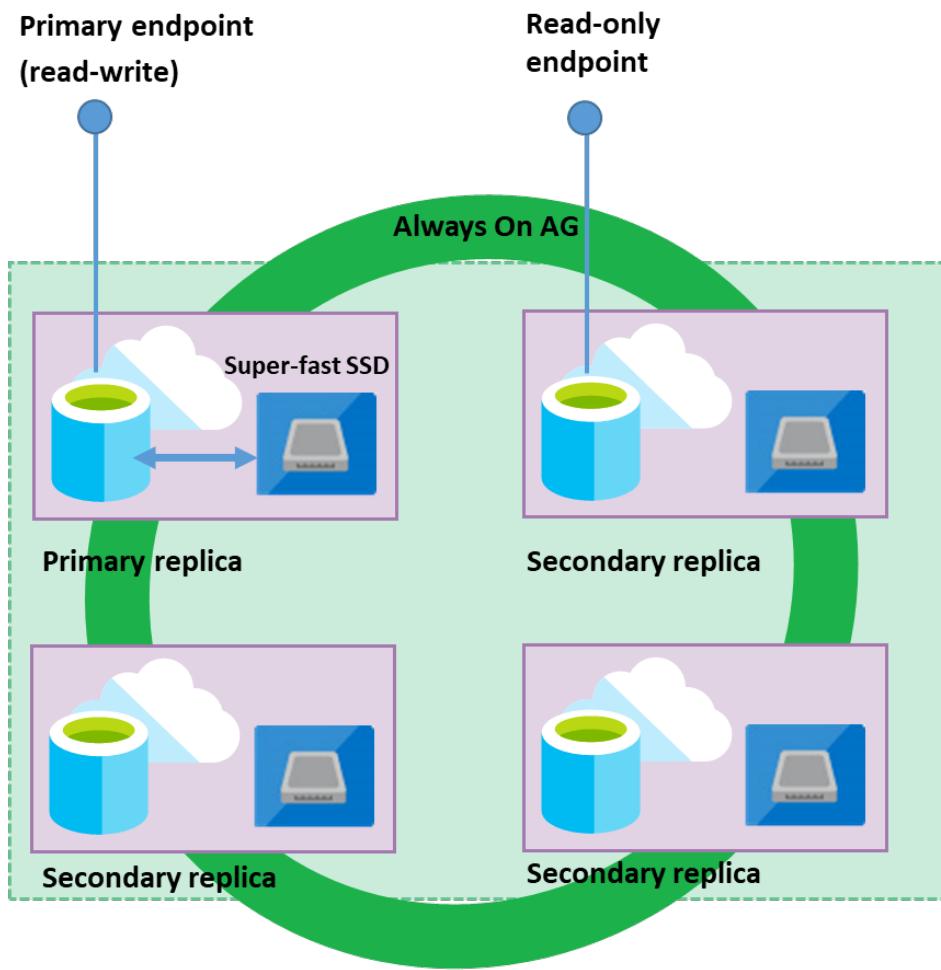
- A stateless compute layer that is running the `sqlserver.exe` process and contains only transient and cached data (for example – plan cache, buffer pool, column store pool). This stateless SQL Server node is operated by Azure Service Fabric that initializes process, controls health of the node, and performs failover to another place if necessary.
- A stateful data layer with database files (.mdf/.ldf) that are stored in Azure Premium Storage. Azure 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 in case of failover 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.

## Premium and Business Critical service tier availability

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 4-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 a 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.

## Zone redundant configuration

By default, the quorum-set replicas for the local storage configurations are created in the same datacenter. With the introduction of [Azure Availability Zones](#), you have the ability to place the different replicas in the quorum-sets 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 does not create additional database redundancy, the use of Availability Zones in the Premium or Business Critical tiers is available at no extra cost. By selecting a zone redundant database, you can make your Premium or Business

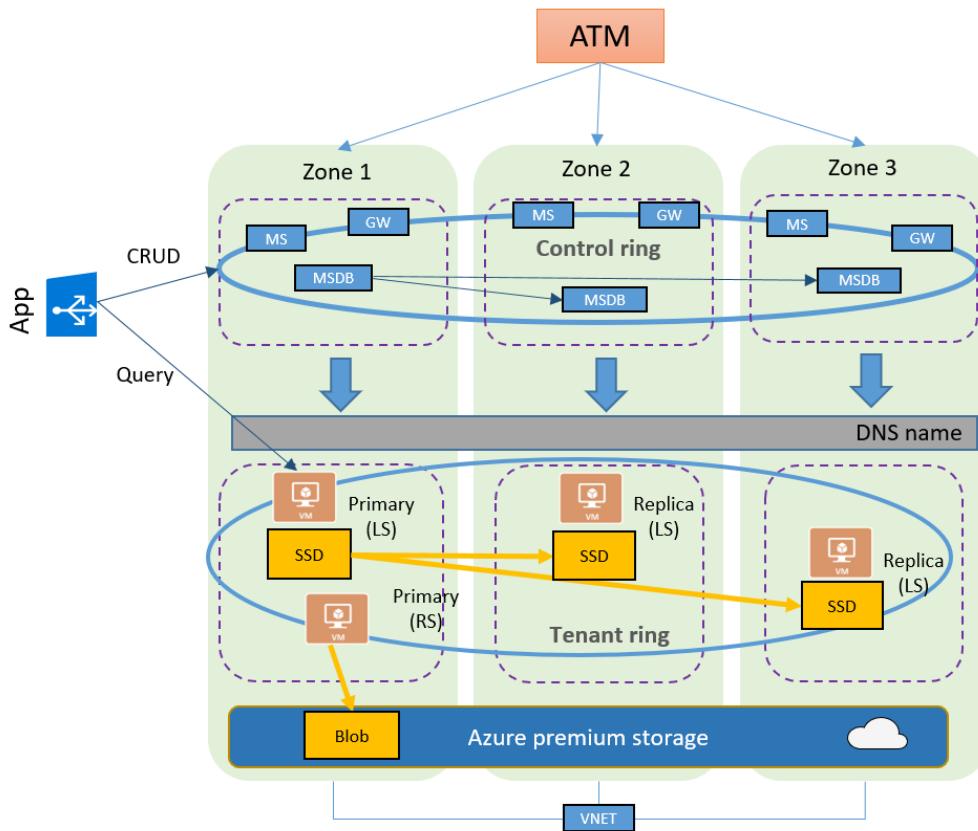
Critical databases resilient to a much larger set of failures, including catastrophic datacenter outages, without any changes of the application logic. You can also convert any existing Premium or Business Critical databases or pools to the zone redundant configuration.

Because the zone redundant quorum-set has 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 a size of data operation and is similar to the regular service tier update. 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 service tier. By default, backups and audit records are stored in RA-GRS storage and therefore may not be automatically available in case of a zone-wide outage.

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, by redesigning the SQL database engine recovery process. ADR is currently available for single databases, elastic pools, and Azure SQL Data Warehouse.

## Conclusion

Azure SQL Database is deeply integrated with the Azure platform and is highly dependent on Service Fabric for failure detection and recovery, on Azure Storage Blobs for data protection and Availability Zones for higher fault tolerance. At the same time, Azure SQL database fully leverages the Always On Availability Group

technology from SQL Server box product for replication and failover. The combination of these technologies enables the 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](#)

# Accelerated Database Recovery (preview)

10/30/2018 • 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, elastic pools, and Azure SQL Data Warehouse. 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.

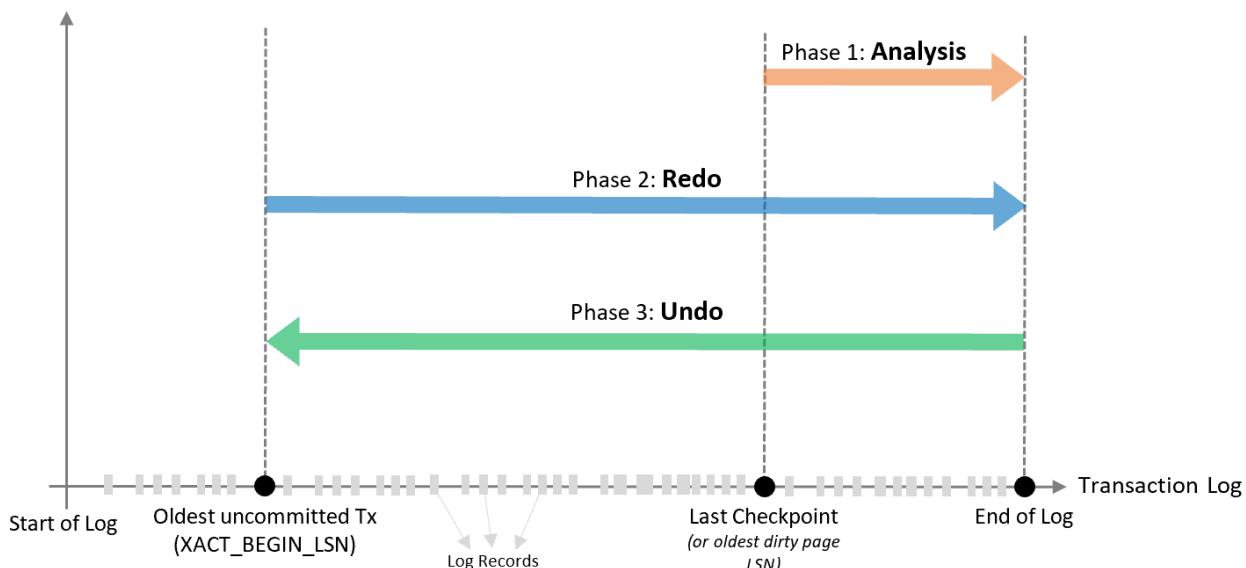
- **Aggressive log truncation**

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.

Recovery Phase / Transaction Log (without ADR)



- **Analysis phase**

Forward scan of the transaction log from the beginning of the last successful checkpoint (or the oldest 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 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 log space.

## The Accelerated Database Recovery process

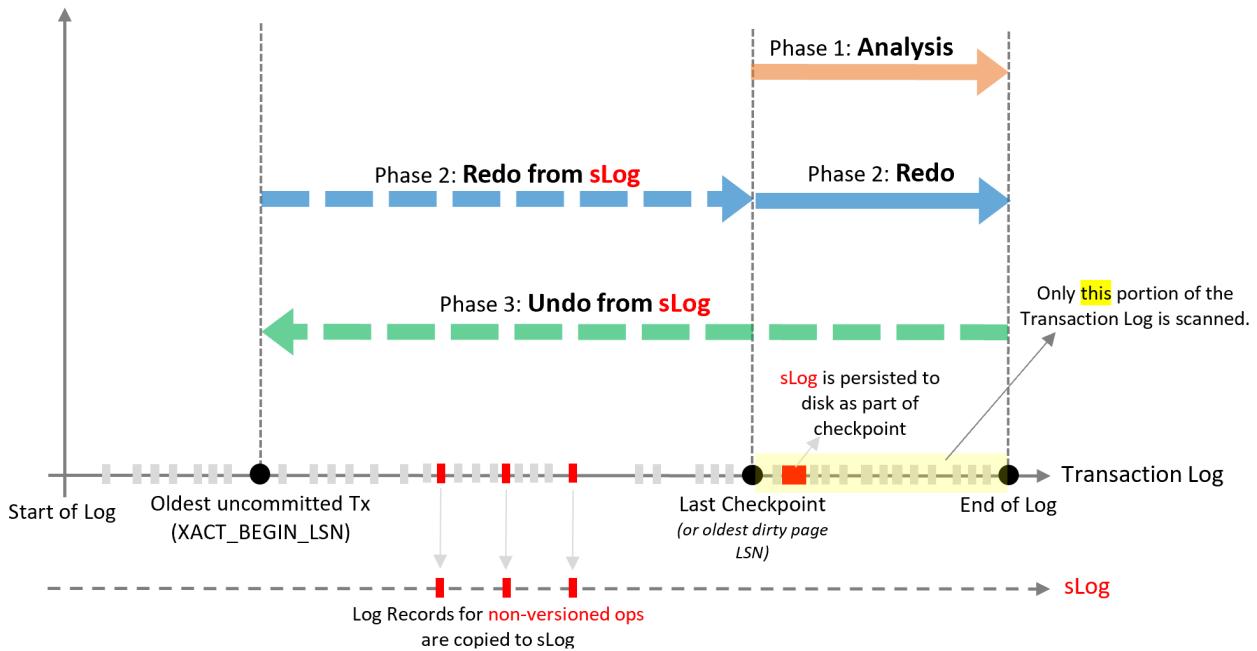
ADR addresses the above issues by completely redesigning the SQL database engine recovery process to:

- 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 ops.

- **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.

- Keeps track of all aborted transactions

- Performs rollback using PVS for all user transactions
- Releases all locks immediately after transaction abort

- **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).

## To enable ADR during this preview period

During the preview period for this feature, send an email to [adr@microsoft.com](mailto:adr@microsoft.com) to learn more and try out Accelerated Database Recovery (ADR). In the e-mail, include the name of your logical server (for single databases, elastic pools, and Azure Data Warehouse). Since this is a preview feature, your testing server should be a non-production server.

# Learn about automatic SQL Database backups

10/24/2018 • 8 minutes to read • [Edit Online](#)

SQL Database automatically creates database backups and uses Azure read-access geo-redundant storage (RA-GRS) to provide geo-redundancy. These backups are created automatically and at no additional charge. You don't need to do anything to make them happen. 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, you can configure a long-term backup retention policy. For more information, see [Long-term retention](#).

## 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](#), [differential](#), and [transaction log](#) backups for the purposes of Point-in-time restore (PITR). The transaction log backups generally occur every 5 - 10 minutes and differential backups generally occur every 12 hours, with the frequency based on the compute size and amount of database activity. Full, differential, and transaction log backups allow you to restore a database to a specific point-in-time to the same server that hosts the database. 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 a database to a point-in-time within the retention period. This operation will create a new database in the same server as the original database.
- Restore a deleted database to the time it was deleted or any time within the retention period. The deleted database can only be restored in the same server where the original database was created.
- Restore a database to another geographical region. This 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 if the database has been configured with a long-term retention policy (LTR). This allows you to restore an old version of the database to satisfy a compliance request or to run an old version of the application. 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.

## How long are backups kept

Each SQL Database backup has a default retention period that is based on the service tier of the database, and

differs between the [DTU-based purchasing model](#) and the [vCore-based purchasing model](#). You can update the backup retention period for a database. See [Change Backup Retention Period](#) for more details.

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 PITR retention period, you can modify the backup properties to add one or more long-term retention periods to your database. See [Long-term backup retention](#) for more details.

#### **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.

## **PITR retention period**

### **DTU-based purchasing model**

The default retention period for a database created using the DTU-based purchasing model depends on the service tier:

- Basic service tier is 1 week.
- Standard service tier is 5 weeks.
- Premium service tier is 5 weeks.

### **vCore-based purchasing model**

If you're using the [vCore-based purchasing model](#), the default backup retention period is 7 days (both on Logical Servers and Managed Instances).

- For single and pooled databases, you can [change backup retention period up to 35 days](#).
- Changing backup retention period is not available in Managed Instance.

If you reduce the current retention period, all existing backups older than the new retention period are no longer be available. If you increase the current retention period, SQL Database will keep the existing backups until the longer retention period is reached.

## How often do backups happen

### **Backups for 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.

The PITR backups are geo-redundant and protected by [Azure Storage cross-regional replication](#)

For more information, see [Point-in-time restore](#)

### **Backups for long-term retention**

SQL Database hosted in Logical Server offers 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 geo-redundant and protected by [Azure Storage cross-regional replication](#).

For more information, see [Long-term backup retention](#).

## Are backups encrypted

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](#).

## How does Microsoft ensure backup integrity

On an ongoing basis, the Azure SQL Database engineering team automatically tests the restore of automated database backups of databases across the service. Upon restore, databases also receive integrity checks using DBCC CHECKDB. 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](#).

## How do automated backups impact my compliance

When you migrate your database from a DTU-based service tier with the default PITR retention of 35 days, 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. See [Change Backup Retention Period](#) for more details.

### 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](#).

## How to change backup retention period

### NOTE

Default backup retention period (7 days) cannot be changed on Managed Instance.

You can change the default retention using REST API or PowerShell. The supported values are: 7, 14, 21, 28 or 35 days. The following examples illustrate how to change PITR retention to 28 days.

### NOTE

These APIs will only impact the PITR retention period. If you configured LTR for your database, it will not be impacted.

See [Long-term backup retention](#) for details of how to change the LTR retention period(s).

## Change PITR backup retention period using PowerShell

```
Set-AzureRmSqlDatabaseBackupShortTermRetentionPolicy -ResourceGroupName resourceGroup -ServerName testserver -DatabaseName testDatabase -RetentionDays 28
```

### IMPORTANT

This API is included in AzureRM.Sql PowerShell Module starting from version [4.7.0-preview](#).

## 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/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/backupShortTermRetentionPolicies/default",  
  "name": "default",  
  "type": "Microsoft.Sql/resourceGroups/servers/databases/backupShortTermRetentionPolicies",  
  "properties": {  
    "retentionDays": 28  
  }  
}
```

See [Backup Retention REST API](#) for more details.

## 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](#).

# Store Azure SQL Database backups for up to 10 years

10/24/2018 • 3 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 [RA-GRS](#) blob storage for up to 10 years. You can then restore any backup as a new database.

## NOTE

LTR can be enabled on the databases hosted in Azure SQL Database Logical Servers. It is not yet available for databases hosted in Managed Instances.

## How SQL Database long-term retention works

Long-term backup retention (LTR) leverages the full database backups that are [automatically created](#) to enable point-in-time restore (PITR). These backups are copied to different storage blobs if LTR policy is configured. You can configure a LTR policy for each SQL database and specify how frequently you need to copy the backups to the long-term storage blobs. To enable that flexibility you can define the 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, one backup during the first week 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. Each backup will be kept in the long-term storage for the period specified by these parameters.

Examples:

- W=0, M=0, Y=5, WeekOfYear=3

The 3rd full backup of each year will be kept for 5 years.

- W=0, M=3, Y=0

The first full backup of each month will be kept for 3 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 6 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 were to modify the above policy and set W=0 (no weekly backups), the cadence of backup copies would change as shown in the above table by the highlighted dates. The storage amount needed to keep these backups would reduce accordingly.

#### NOTE

1. The LTR copies are created by Azure storage service so the copy process has no performance impact on the existing database.
2. The policy applies to the future backups. E.g. if the specified WeekOfYear is in the past when the policy is configured, the first LTR backup will be created next year.
3. 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.

## 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. This will not

increase your LTR storage cost as backups are not generated from the secondaries. Only when the secondary becomes primary the backups will be created. This way you will guarantee 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 the outage that cause it to 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 using PowerShell, see [Configure 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](#).

# Manage Azure SQL Database long-term backup retention

10/26/2018 • 5 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 backups in Azure blob storage 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.

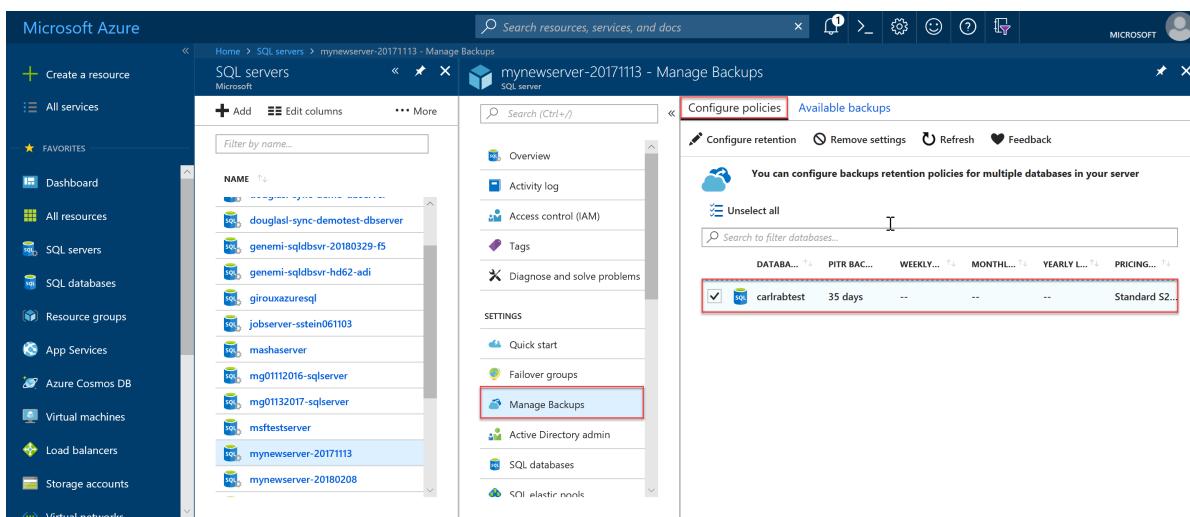
## Use the Azure portal to configure long-term retention policies and restore backups

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.

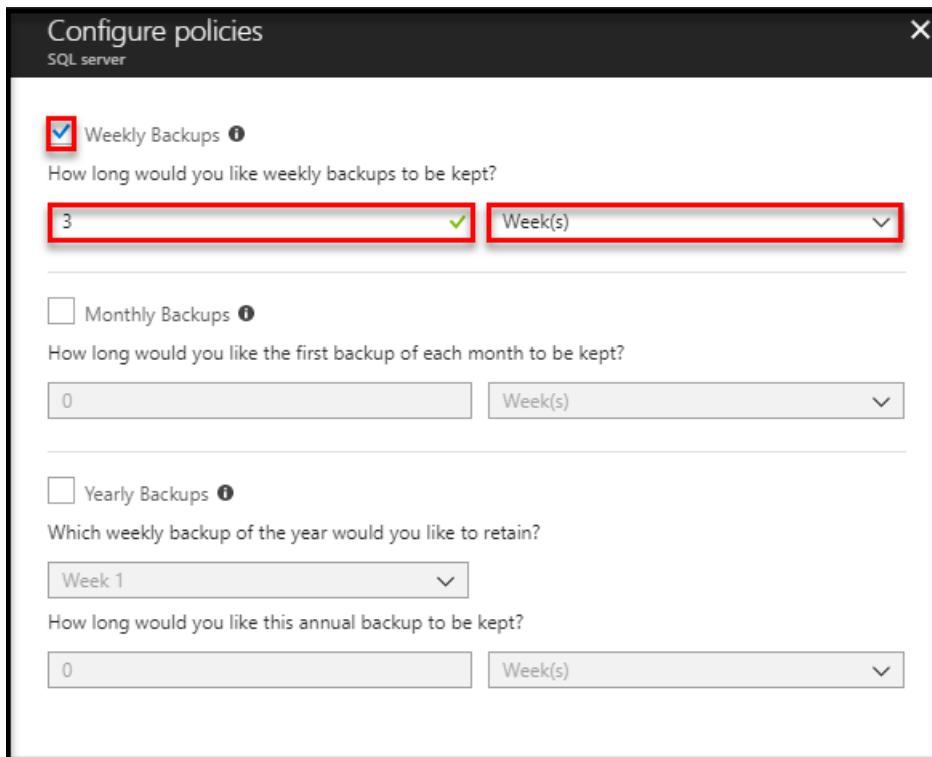
### Configure long-term retention policies

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.



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**.

#### View backups and restore from a backup using Azure portal

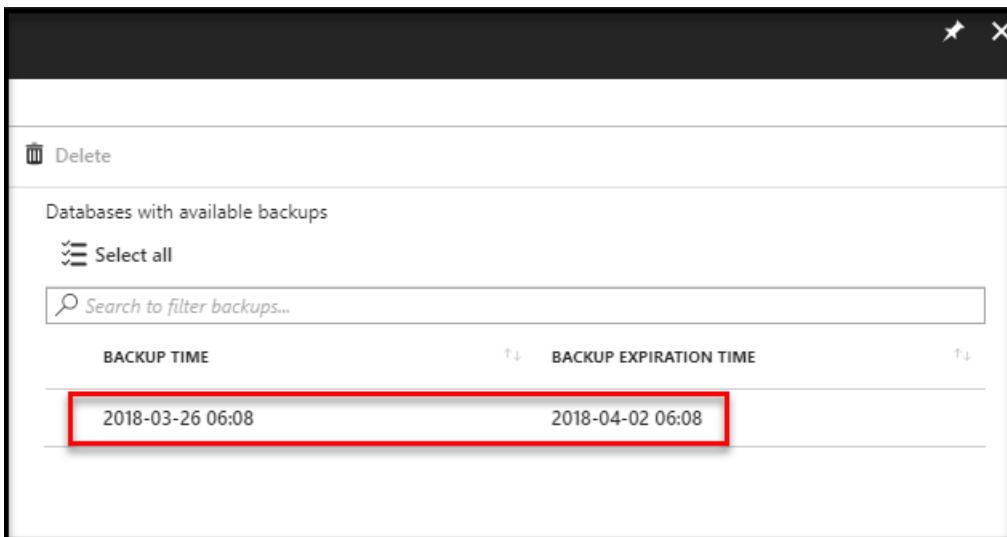
View the backups that are retained for a specific database with a LTR policy, and restore from those backups.

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.

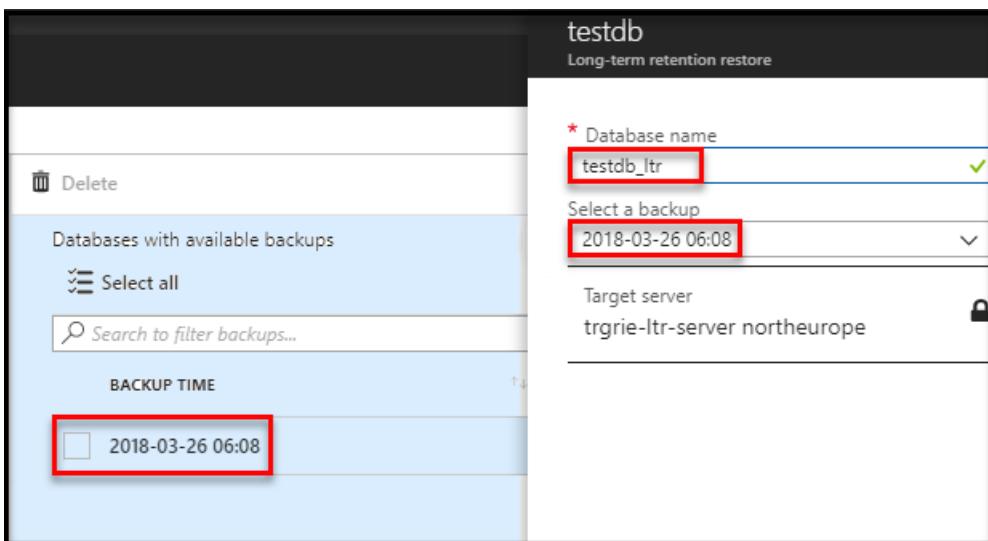
The screenshot shows the 'Available backups' tab in the Azure portal. It displays a list of databases with available backups. The database 'testdb' is highlighted with a red box. The table has columns: DATABASE, DATABASE DROPPED TIME, and SERVER NAME. The 'testdb' row shows the database name, a dropped time of '--', and the server name 'trgrie-ltr-server'. Other rows listed are 'testdb\_2018-03-31T10-40Z', 'testdb\_2018-03-26T13-08Z\_newsku', and 'testdb\_2018-03-26T13-08Z\_pool2'.

| 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.

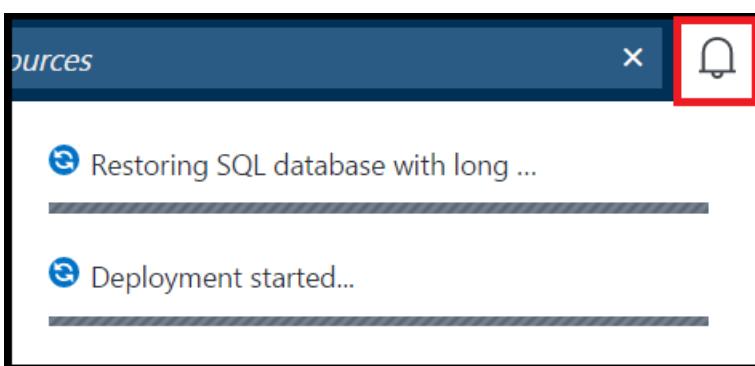


3. Select the backup from which you want to restore, and then specify the new database name.



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.



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.

# Use PowerShell to configure long-term retention policies and restore backups

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.

## IMPORTANT

LTR V2 API is supported in the following PowerShell versions:

- [AzureRM.Sql-4.5.0](#) or newer
- [AzureRM-6.1.0](#) or newer

## RBAC roles to manage long-term retention

In order to manage LTR backups, you will need to be

- Subscription Owner or
- SQL Server Contributor role in **Subscription** scope or
- SQL Database Contributor role in **Subscription** scope

If more granular control is required, you can create custom RBAC roles and assign them in **Subscription** scope.

For **Get-AzureRmSqlDatabaseLongTermRetentionBackup** and **Restore-AzureRmSqlDatabase** the role needs to have following permissions:

Microsoft.Sql/locations/longTermRetentionBackups/read  
Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionBackups/read  
Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/read

For **Remove-AzureRmSqlDatabaseLongTermRetentionBackup** the role need to have following permissions:

Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/delete

## Create an LTR policy

```
# Get the SQL server
# $subId = "{subscription-id}"
# $serverName = "{server-name}"
# $resourceGroup = "{resource-group-name}"
# $dbName = "{database-name}"

Connect-AzureRmAccount
Select-AzureRmSubscription -SubscriptionId $subId

# get the server
$server = Get-AzureRmSqlServer -ServerName $serverName -ResourceGroupName $resourceGroup

# create LTR policy with WeeklyRetention = 12 weeks. MonthlyRetention and YearlyRetention = 0 by default.
Set-AzureRmSqlDatabaseBackupLongTermRetentionPolicy -ServerName $serverName -DatabaseName $dbName -
ResourceGroupName $resourceGroup -WeeklyRetention P12W

# create LTR policy with WeeklyRetention = 12 weeks, YearlyRetetion = 5 years and WeekOfYear = 16 (week of April 15). MonthlyRetention = 0 by default.
Set-AzureRmSqlDatabaseBackupLongTermRetentionPolicy -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-AzureRmSqlDatabase -ResourceGroupName Default-SQL-WestCentralUS -ServerName trgrie-ltr-
server | Get-AzureRmSqlDatabaseLongTermRetentionPolicy -Current

# Get the LTR policy of a specific database
$ltrPolicies = Get-AzureRmSqlDatabaseBackupLongTermRetentionPolicy -ServerName $serverName -DatabaseName
$dbName -ResourceGroupName $resourceGroup -Current
```

## Clear an LTR policy

This example shows how to clear an LTR policy from a database

```
Set-AzureRmSqlDatabaseBackupLongTermRetentionPolicy -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
# The backups are grouped by the logical database id.
# Within each group they are ordered by the timestamp, the earliest
# backup first.
$ltrBackups = Get-AzureRmSqlDatabaseLongTermRetentionBackup -Location $server.Location

# Get the list of LTR backups from the Azure region under
# the named server.
$ltrBackups = Get-AzureRmSqlDatabaseLongTermRetentionBackup -Location $server.Location -ServerName $serverName

# Get the LTR backups for a specific database from the Azure region under the named server
$ltrBackups = Get-AzureRmSqlDatabaseLongTermRetentionBackup -Location $server.Location -ServerName $serverName
-DatabaseName $dbName

# List LTR backups only from live databases (you have option to choose All/Live/Deleted)
$ltrBackups = Get-AzureRmSqlDatabaseLongTermRetentionBackup -Location $server.Location -DatabaseState Live

# Only list the latest LTR backup for each database
$ltrBackups = Get-AzureRmSqlDatabaseLongTermRetentionBackup -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-AzureRmSqlDatabaseLongTermRetentionBackup -ResourceId $ltrBackup.ResourceId
```

### IMPORTANT

Deleting LTR backup is non-reversible. 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 LTR backup as an S3 database
Restore-AzureRmSqlDatabase -FromLongTermRetentionBackup -ResourceId $ltrBackup.ResourceId -ServerName
$serverName -ResourceGroupName $resourceGroup -TargetDatabaseName $dbName -ServiceObjectiveName S3
```

#### 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](#)

# Configure long-term backup retention using Azure Recovery Services Vault

10/8/2018 • 8 minutes to read • [Edit Online](#)

You can configure the Azure Recovery Services vault to store Azure SQL database backups and then recover a database using backups retained in the vault using the Azure portal or PowerShell.

## NOTE

As part of the initial release of the preview of long-term backup retention in October 2016, backups were stored in the Azure Services Recovery Service vault. This update removes this dependency, but for backward compatibility the original API is supported until May 31, 2018. If you need to interact with backups in the Azure Services Recovery vault, see [Long-term backup retention using Azure Services Recovery Service vault](#).

## Azure portal

The following sections show you how to use the Azure portal to configure the Azure Recovery Services vault, view backups in the vault, and restore from the vault.

### Configure the vault, register the server, and select databases

You configure an Azure Recovery Services vault to [retain automated backups](#) for a period longer than the retention period for your service tier.

1. Open the **SQL Server** page for your server.

sqltutorialserver

SQL server

+ New pool Import database Reset password Delete Move

Search (Ctrl+)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Quick start

Firewall

Long-term backup retention

Auditing & Threat detection

Latest SQL database update

Active Directory admin

Properties

Locks

Automation script

SUPPORT + TROUBLESHOOTING

Automatic tuning

New support request

Resource group  
sqltutorialrg

Status  
Available

Location  
West US

Subscription name

Server version  
V12

Auditing  
Not configured

Server admin  
sqladmin

Active Directory admin  
Not configured

Firewall  
Show firewall settings

SQL databases

0 Databases

| DATABASE           | STATUS | PRICING TIER |
|--------------------|--------|--------------|
| No databases found |        |              |

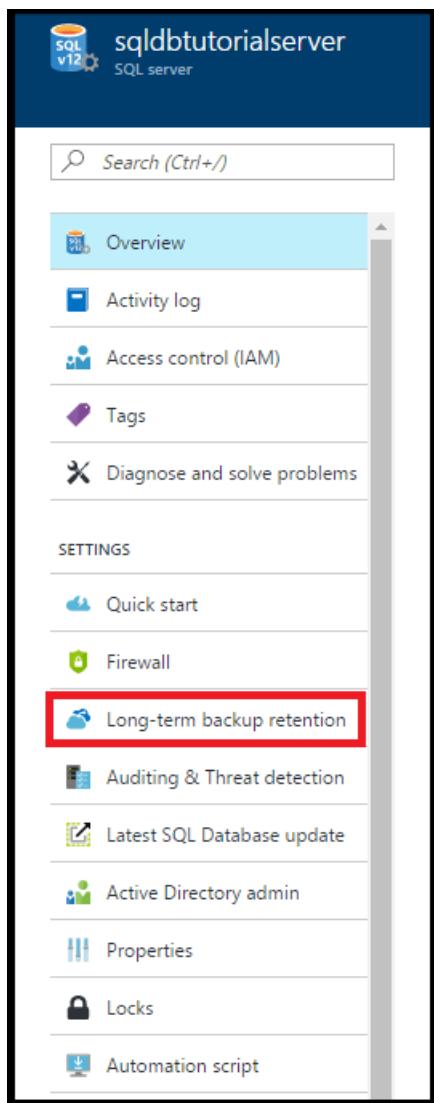
Elastic database pools

Elastic database pools

0 Elastic database pools

| NAME                   | PRICING TIER | POOL EDTU |
|------------------------|--------------|-----------|
| No elastic pools found |              |           |

2. Click **Long-term backup retention**.



3. On the **Long-term backup retention** page for your server, review and accept the preview terms (unless you have already done so - or this feature is no longer in preview).

A screenshot of the 'Long-term backup retention' configuration page for the 'sqldbtutorialserver'. The top navigation bar includes Save, Discard, Configure, and Disable buttons. A descriptive text box states: 'Long-term backup retention allows you to store weekly backups in Azure recovery services vault up to 10 years. This helps you meet compliance and other data retention requirements. Once configured, backups show up in the vault within next 7 days.' Below this, a message box says 'Preview terms: Not Accepted. To sign up for preview, click here.' A note below it says 'Recovery service vault : Not configured'. The main area shows a table of databases with their current retention settings. The table has columns: DATABASE, POLICY NAME, RETENTION PERIOD, and PRICING TIER.

| DATABASE              | POLICY NAME | RETENTION PERIOD | PRICING TIER |
|-----------------------|-------------|------------------|--------------|
| sqldbtutorialdb       | None        | None             | Basic        |
| sqldbtutorialdb_20... | None        | None             | Basic        |
| blankdb               | None        | None             | Basic        |

4. To configure long-term backup retention, select that database in the grid and then click **Configure** on the toolbar.

The screenshot shows the 'Long-term backup retention' configuration page for a SQL server named 'sqldbtutorialserver'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Firewall, Long-term backup retention (which is selected and highlighted in blue), and Auditing & Threat detection. The main content area has a heading about long-term backup retention and a note that the recovery service vault is not configured. It shows a table of databases with their current retention policies. The database 'sqldbtutorialdb' is selected and highlighted with a red border.

| DATABASE              | POLICY NAME | RETENTION PERIOD | PRICING TIER |
|-----------------------|-------------|------------------|--------------|
| blankdb               | None        | None             | Basic        |
| sqldbtutorialdb       | None        | None             | Basic        |
| sqldbtutorialdb_20... | None        | None             | Basic        |

5. On the **Configure** page, click **Configure required settings** under **Recovery service vault**.

The screenshot shows the 'Configure' page with the 'Recovery service vault' section highlighted. It includes fields for selecting a vault and creating a new policy. A retention policy named '1 Year(s)' is selected.

6. On the **Recovery services vault** page, select an existing vault, if any. Otherwise, if no recovery services vault found for your subscription, click to exit the flow and create a recovery services vault.

The screenshot shows the 'Recovery services vault' page with a message indicating that no recovery services vault was found. It provides a link to exit the flow and create a new vault.

7. On the **Recovery Services vaults** page, click **Add**.

**Recovery Services vaults**

Microsoft

**+ Add**   **Columns**   **Refresh**

**Subscriptions:** 1 of 2 selected

Filter by name...

0 items

| NAME   | RESOURCE GROUP | LOCATION | SUBSCRIPTION |
|--|----------------|----------|--------------|
| No Recovery Services vaults to display in the selected subscriptions |                |          |              |

8. On the **Recovery Services vault** page, provide a valid name for the Recovery Services vault.

**Recovery Services vault**

Recovery Services vault

\* Name  ✓

\* Subscription

\* Resource group ⓘ  
 Create new  Use existing  
Select

\* Location  ▼

Pin to dashboard

**Create**   Automation options

9. Select your subscription and resource group, and then select the location for the vault. When done, click **Create**.

Recovery Services vault

Name: sqldbtutorialvault

Subscription: [dropdown]

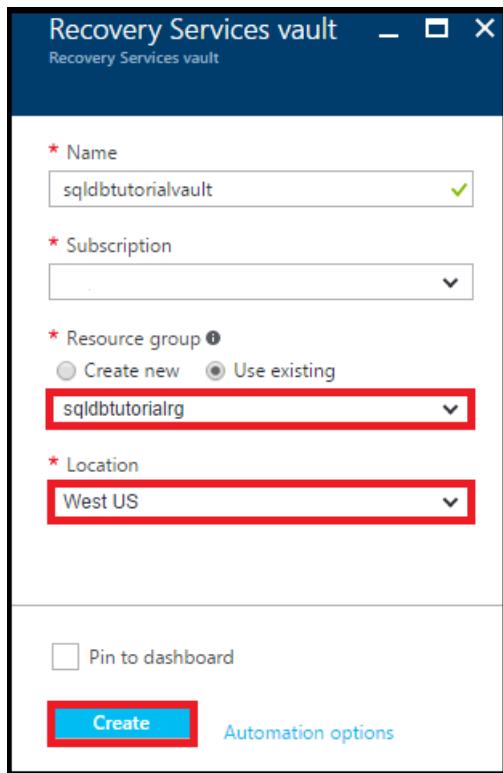
Resource group:  Use existing  
sqldbtutorialrg

Location: West US

Pin to dashboard

**Create**

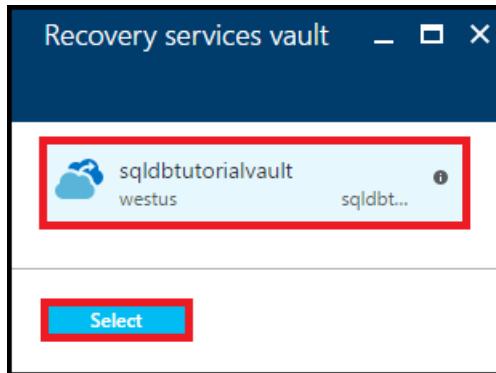
Automation options



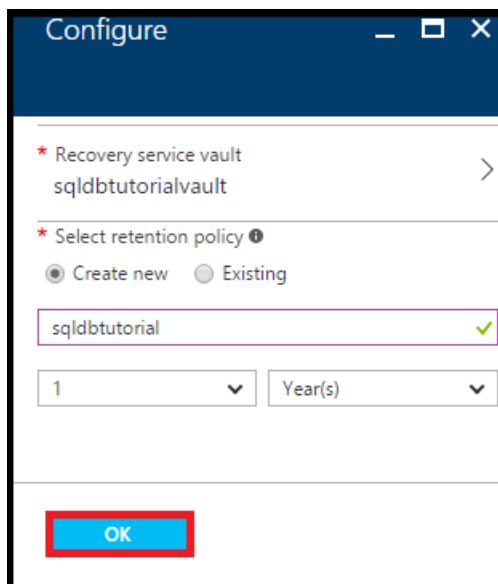
**IMPORTANT**

The vault must be located in the same region as the Azure SQL logical server, and must use the same resource group as the logical server.

10. After the new vault is created, execute the necessary steps to return to the **Recovery services vault** page.
11. On the **Recovery services vault** page, click the vault and then click **Select**.



12. On the **Configure** page, provide a valid name for the new retention policy, modify the default retention policy as appropriate, and then click **OK**.



#### NOTE

Retention policy names don't allow some characters, including spaces.

13. On the **Long-term backup retention** page for your database, click **Save** and then click **OK** to apply the long-term backup retention policy to all selected databases.



14. Click **Save** to enable long-term backup retention using this new policy to the Azure Recovery Services vault that you configured.

Long-term backup retention allows you to store weekly backups in Azure recovery services vault up to 10 years. This helps you meet compliance and other data retention requirements. Once configured, backups show up in the vault within next 7 days.

Preview terms: Accepted. To read the terms, click here.

| DATABASE  | POLICY NAME   | RETENTION PERIOD | PRICING TIER |
|---|---------------|------------------|--------------|
| sqldbtutorialdb_20...                               | None          | None             | Basic        |
| blankdb   | None          | None             | Basic        |
| <input checked="" type="checkbox"/> sqldbtutorialdb | sqldbtutorial | 1 Years          | Basic        |

#### IMPORTANT

Once configured, backups show up in the vault within next seven days. Do not continue this tutorial until backups show up in the vault.

## View backups in long-term retention using Azure portal

View information about your database backups in [long-term backup retention](#).

1. In the Azure portal, open your Azure Recovery Services vault for your database backups (go to **All resources** and select it from the list of resources for your subscription) to view the amount of storage used by your database backups in the vault.

The screenshot shows the 'Essentials' blade of an Azure Recovery Services vault. It includes sections for Resource group, Status, Location, Subscription name, and Subscription ID. The 'Monitoring' section displays Backup Alerts (0 Critical, 0 Warning) and Site Recovery Health (0 unhealthy services, 0 events, 0 updates available). The 'Backup' section shows Backup Items (0 Azure Virtual Machines, 0 File-Folders) and Backup Usage (Cloud - LRS: 0 B, Cloud - GRS: 1.75 MB). The 'Site Recovery' section shows Replicated items (0) and Recovery plans (0).

| Backup Items         |   | Backup Usage | Backup Jobs |             |   |
|----------------------|---|--------------|-------------|-------------|---|
| Azure Virtual Mac... | 0 | Cloud - LRS  | 0 B         | In progress | 0 |
| File-Folders         | 0 | Cloud - GRS  | 1.75 MB     | Failed      | 0 |

| Replicated items |   | Recovery plans | Site Recovery jobs |                   |   |
|------------------|---|----------------|--------------------|-------------------|---|
| 0                | 💻 | 0              | 💻                  | Failed            | 0 |
|                  |   |                |                    | Waiting for input | 0 |
|                  |   |                |                    | In progress       | 0 |

2. Open the **SQL database** page for your database.

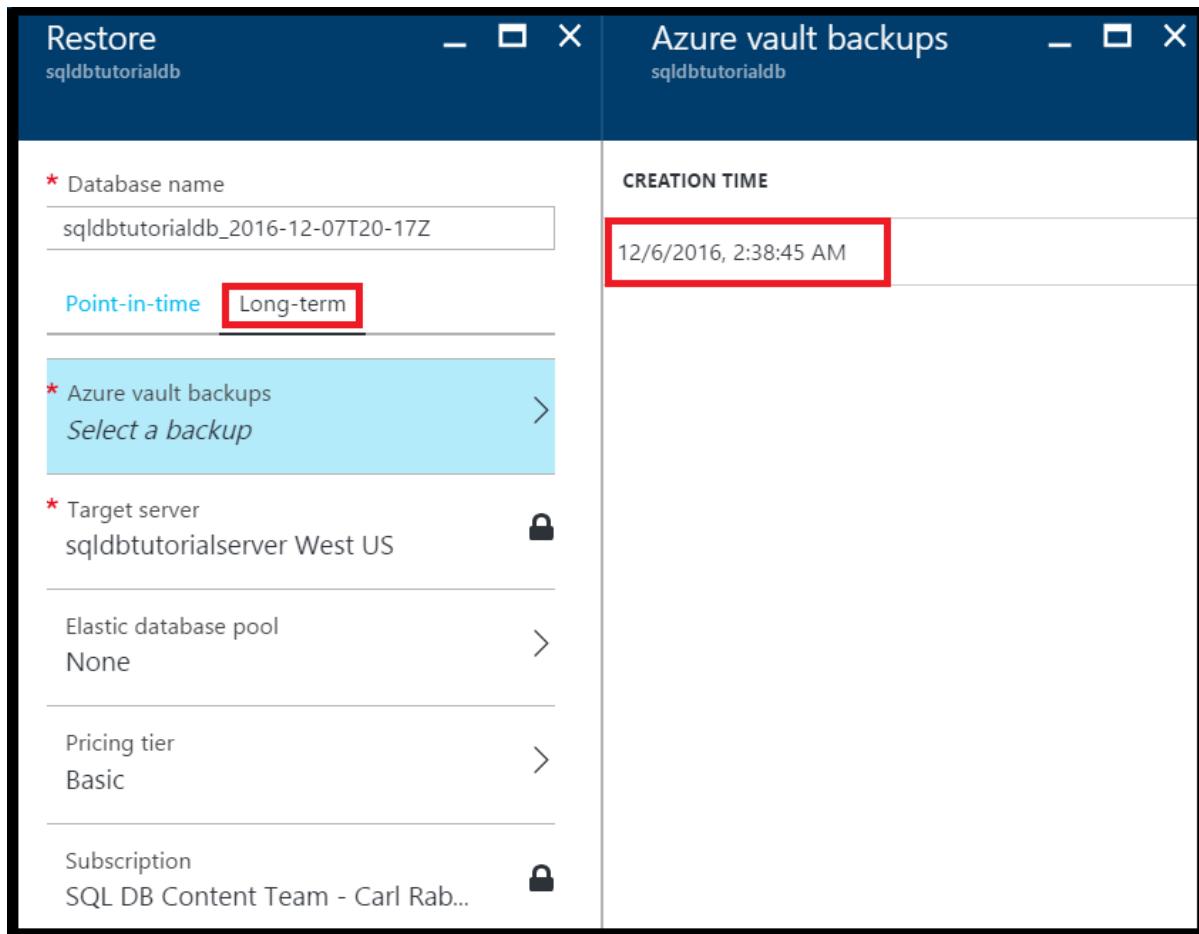
The screenshot shows the Azure portal interface for a SQL database named 'sqldbtutorialdb'. The top navigation bar includes 'Tools', 'Copy', 'Restore', 'Export', 'Set server firew...', and 'Delete'. A search bar at the top left contains the placeholder 'Search (Ctrl+I)'. On the left, a sidebar lists 'Overview', 'Activity log', 'Tags', 'Diagnose and solve problems', 'Quick start', 'Pricing tier (scale DTUs)', 'Geo-Replication', 'Auditing & Threat detection', 'Dynamic data masking', 'Transparent data encryption', 'Properties', 'Locks', 'Automation script', 'Alert rules', and 'Database size'. The main content area is divided into sections: 'Essentials' (Resource group: sqldbtutorialrg, Status: Online, Location: West US, Subscription name: [redacted], Subscription ID: [redacted], Server name: sqldbtutorialserver.database.windows.net, Server version: V12, Connection strings: Show database connection strings, Pricing tier: Basic (5 DTUs), Geo-Replication role: Not configured), 'Monitoring' (Resource utilization chart showing DTU percentage at 4.99% from 2:45 PM to 3:30 PM), 'Operations', and 'Performance'.

3. On the toolbar, click **Restore**.



4. On the Restore page, click **Long-term**.

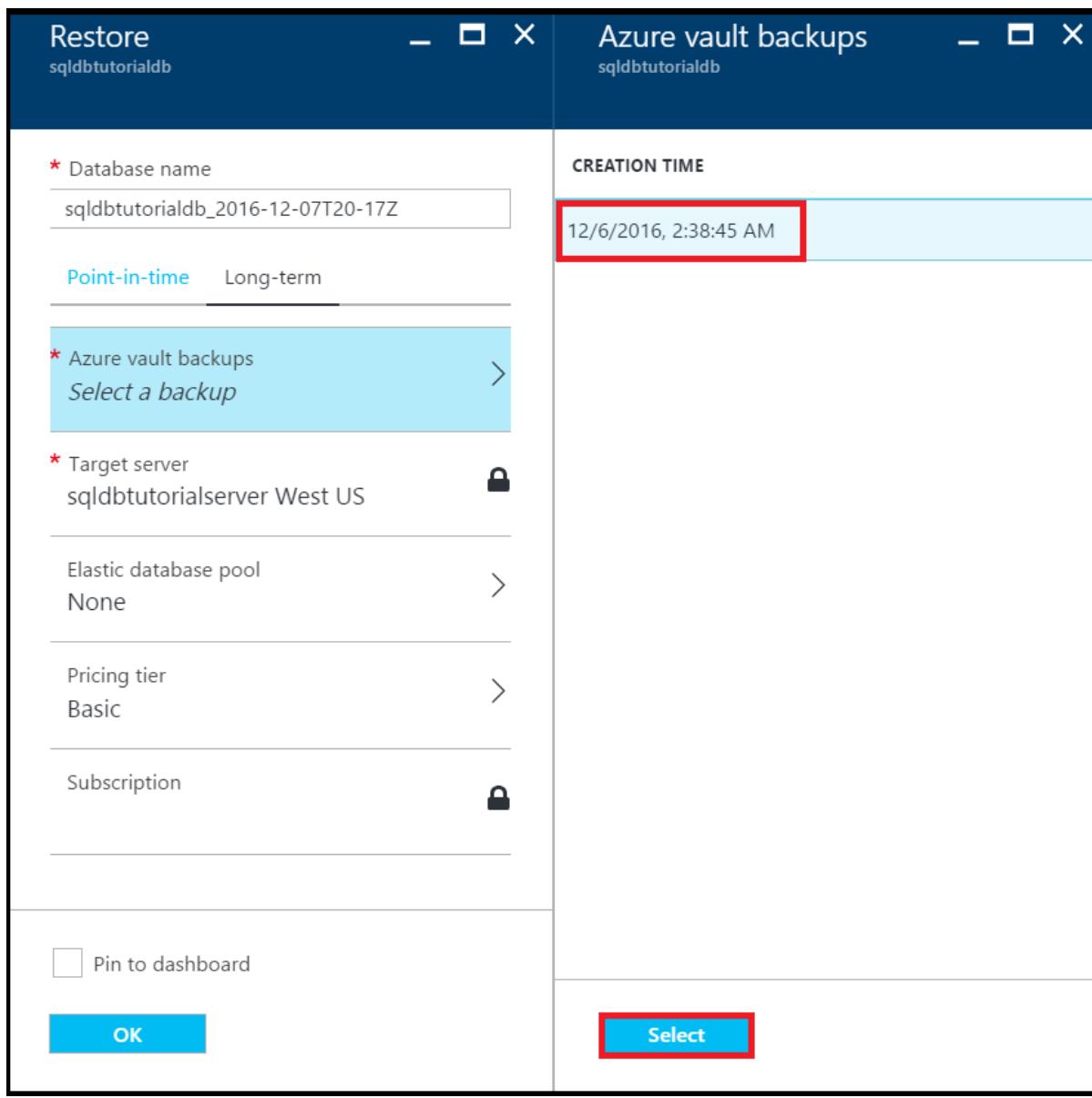
5. Under Azure vault backups, click **Select a backup** to view the available database backups in long-term backup retention.



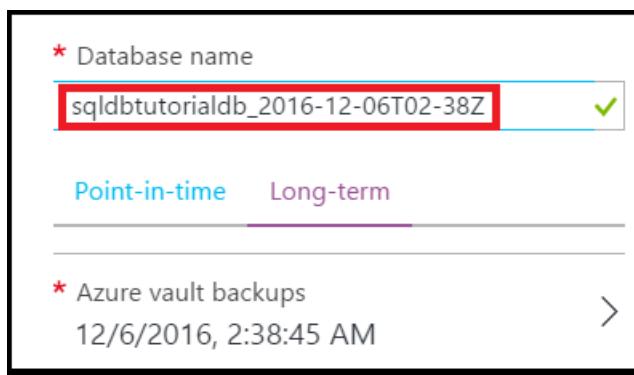
### Restore a database from a backup in long-term backup retention using the Azure portal

You restore the database to a new database from a backup in the Azure Recovery Services vault.

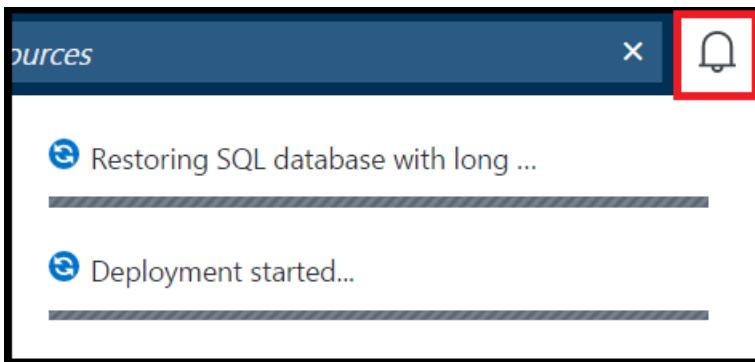
1. On the **Azure vault backups** page, click the backup to restore and then click **Select**.



2. In the **Database name** text box, provide the name for the restored database.



3. Click **OK** to restore your database from the backup in the vault to the new database.
4. On the toolbar, click the notification icon to view the status of the restore job.



- When the restore job is completed, open the **SQL databases** page to view the newly restored database.

**SQL databases**

Subscriptions: 1 of 2 selected

| NAME                              | STATUS | REPLICATION ROLE | SERVER            | PRICING TIER | LOCATION | SUBSCRIPTION            |
|-----------------------------------|--------|------------------|-------------------|--------------|----------|-------------------------|
| blankdb                           | Online | None             | sqltutorialserver | Basic        | West US  | SQL DB Content Team ... |
| sqldbtutorialdb                   | Online | None             | sqltutorialserver | Basic        | West US  | SQL DB Content Team ... |
| sqldbtutorialdb_2016-12-02T09-00Z | Online | None             | sqltutorialserver | Basic        | West US  | SQL DB Content Team ... |
| sqldbtutorialdb_2016-12-07T20-17Z | Online | None             | sqltutorialserver | Basic        | West US  | SQL DB Content Team ... |

#### 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](#).

## PowerShell

The following sections show you how to use PowerShell to configure the Azure Recovery Services vault, view backups in the vault, and restore from the vault.

### Create a recovery services vault

Use the [New-AzureRmRecoveryServicesVault](#) to create a recovery services vault.

#### IMPORTANT

The vault must be located in the same region as the Azure SQL logical server, and must use the same resource group as the logical server.

```
# Create a recovery services vault

#$resourceGroupName = "{resource-group-name}"
#$serverName = "{server-name}"
$serverLocation = (Get-AzureRmSqlServer -ServerName $serverName -ResourceGroupName $resourceGroupName).Location
$recoveryServiceVaultName = "{new-vault-name}"

$vault = New-AzureRmRecoveryServicesVault -Name $recoveryServiceVaultName -ResourceGroupName $ResourceGroupName
-Location $serverLocation
Set-AzureRmRecoveryServicesBackupProperties -BackupStorageRedundancy LocallyRedundant -Vault $vault
```

### Set your server to use the recovery vault for its long-term retention backups

Use the [Set-AzureRmSqlServerBackupLongTermRetentionVault](#) cmdlet to associate a previously created recovery

services vault with a specific Azure SQL server.

```
# Set your server to use the vault to for long-term backup retention

Set-AzureRmSqlServerBackupLongTermRetentionVault -ResourceGroupName $resourceGroupName -ServerName $serverName
-ResourceId $vault.Id
```

## Create a retention policy

A retention policy is where you set how long to keep a database backup. Use the [Get-AzureRmRecoveryServicesBackupRetentionPolicyObject](#) cmdlet to get the default retention policy to use as the template for creating policies. In this template, the retention period is set for 2 years. Next, run the [New-AzureRmRecoveryServicesBackupProtectionPolicy](#) to finally create the policy.

### NOTE

Some cmdlets require that you set the vault context before running ([Set-AzureRmRecoveryServicesVaultContext](#)) so you see this cmdlet in a few related snippets. You set the context because the policy is part of the vault. You can create multiple retention policies for each vault and then apply the desired policy to specific databases.

```
# Retrieve the default retention policy for the AzureSQLDatabase workload type
$retentionPolicy = Get-AzureRmRecoveryServicesBackupRetentionPolicyObject -WorkloadType AzureSQLDatabase

# Set the retention value to two years (you can set to any time between 1 week and 10 years)
$retentionPolicy.RetentionDurationType = "Years"
$retentionPolicy.RetentionCount = 2
$retentionPolicyName = "my2YearRetentionPolicy"

# Set the vault context to the vault you are creating the policy for
Set-AzureRmRecoveryServicesVaultContext -Vault $vault

# Create the new policy
$policy = New-AzureRmRecoveryServicesBackupProtectionPolicy -name $retentionPolicyName -WorkloadType
AzureSQLDatabase -retentionPolicy $retentionPolicy
$policy
```

## Configure a database to use the previously defined retention policy

Use the [Set-AzureRmSqlDatabaseBackupLongTermRetentionPolicy](#) cmdlet to apply the new policy to a specific database.

```
# Enable long-term retention for a specific SQL database
$policyState = "enabled"
Set-AzureRmSqlDatabaseBackupLongTermRetentionPolicy -ResourceGroupName $resourceGroupName -ServerName
$serverName -DatabaseName $databaseName -State $policyState -ResourceId $policy.Id
```

## View backup info, and backups in long-term retention

View information about your database backups in [long-term backup retention](#).

Use the following cmdlets to view backup information:

- [Get-AzureRmRecoveryServicesBackupContainer](#)
- [Get-AzureRmRecoveryServicesBackupItem](#)
- [Get-AzureRmRecoveryServicesBackupRecoveryPoint](#)

```

##$resourceGroupName = "{resource-group-name}"
##$serverName = "{server-name}"
$databasenameedingRestore = $databaseName

# Set the vault context to the vault we want to restore from
#$vault = Get-AzureRmRecoveryServicesVault -ResourceGroupName $resourceGroupName
Set-AzureRmRecoveryServicesVaultContext -Vault $vault

# the following commands find the container associated with the server 'myserver' under resource group
'myresourcegroup'
$container = Get-AzureRmRecoveryServicesBackupContainer -ContainerType AzureSQL -FriendlyName $vault.Name

# Get the long-term retention metadata associated with a specific database
$item = Get-AzureRmRecoveryServicesBackupItem -Container $container -WorkloadType AzureSQLDatabase -Name
$databasenameedingRestore

# Get all available backups for the previously indicated database
# Optionally, set the -StartDate and -EndDate parameters to return backups within a specific time period
$availableBackups = Get-AzureRmRecoveryServicesBackupRecoveryPoint -Item $item
$availableBackups

```

## Restore a database from a backup in long-term backup retention

Restoring from long-term backup retention uses the [Restore-AzureRmSqlDatabase](#) cmdlet.

```

# Restore the most recent backup: $availableBackups[0]
##$resourceGroupName = "{resource-group-name}"
##$serverName = "{server-name}"
$restoredDatabaseName = "{new-database-name}"
$edition = "Basic"
$performanceLevel = "Basic"

$restoredDb = Restore-AzureRmSqlDatabase -FromLongTermRetentionBackup -ResourceId $availableBackups[0].Id -
ResourceGroupName $resourceGroupName ` 
-ServerName $serverName -TargetDatabaseName $restoredDatabaseName -Edition $edition -ServiceObjectiveName
$performanceLevel
$restoredDb

```

### 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](#).

## How to cleanup backups in Recovery Services vault

As of July 1, 2018 the LTR V1 API has been deprecated and all your existing backups in Recovery Service vaults have been migrated to the LTR storage containers managed by SQL Database. To ensure that you are no longer charged for the original backups, they have been removed from the vaults after migration. However, if you placed a lock on your vault the backups will remain there. To avoid the unnecessary charges, you can manually remove the old backups from the Recovery Service vault using the following script.

```

<#
.EXAMPLE
.\Drop-LtrV1Backup.ps1 -SubscriptionId "{vault_sub_id}" -ResourceGroup "{vault_resource_group}" -VaultName
"{vault_name}"
#>
[CmdletBinding()]
Param (
    [Parameter(Mandatory = $true, HelpMessage="The vault subscription ID")]
    $SubscriptionId,
    [Parameter(Mandatory = $true, HelpMessage="The vault resource group name")]
    $ResourceGroup,
    [Parameter(Mandatory = $true, HelpMessage="The vault name")]
    $VaultName
)
Login-AzureRmAccount
Select-AzureRmSubscription -SubscriptionId $SubscriptionId
$vaults = Get-AzureRmRecoveryServicesVault
$vault = $vaults | where { $_.Name -eq $VaultName }
Set-AzureRmRecoveryServicesVaultContext -Vault $vault
$containers = Get-AzureRmRecoveryServicesBackupContainer -ContainerType AzureSQL
ForEach ($container in $containers)
{
    $canDeleteContainer = $true
    $ItemCount = 0
    Write-Host "Working on container" $container.Name
    $items = Get-AzureRmRecoveryServicesBackupItem -container $container -WorkloadType AzureSQLDatabase
    ForEach ($item in $items)
    {
        write-host "Deleting item" $item.name
        Disable-AzureRmRecoveryServicesBackupProtection -RemoveRecoveryPoints -item $item -Force
    }
    Write-Host "Deleting container" $container.Name
    Unregister-AzureRmRecoveryServicesBackupContainer -Container $container
}

```

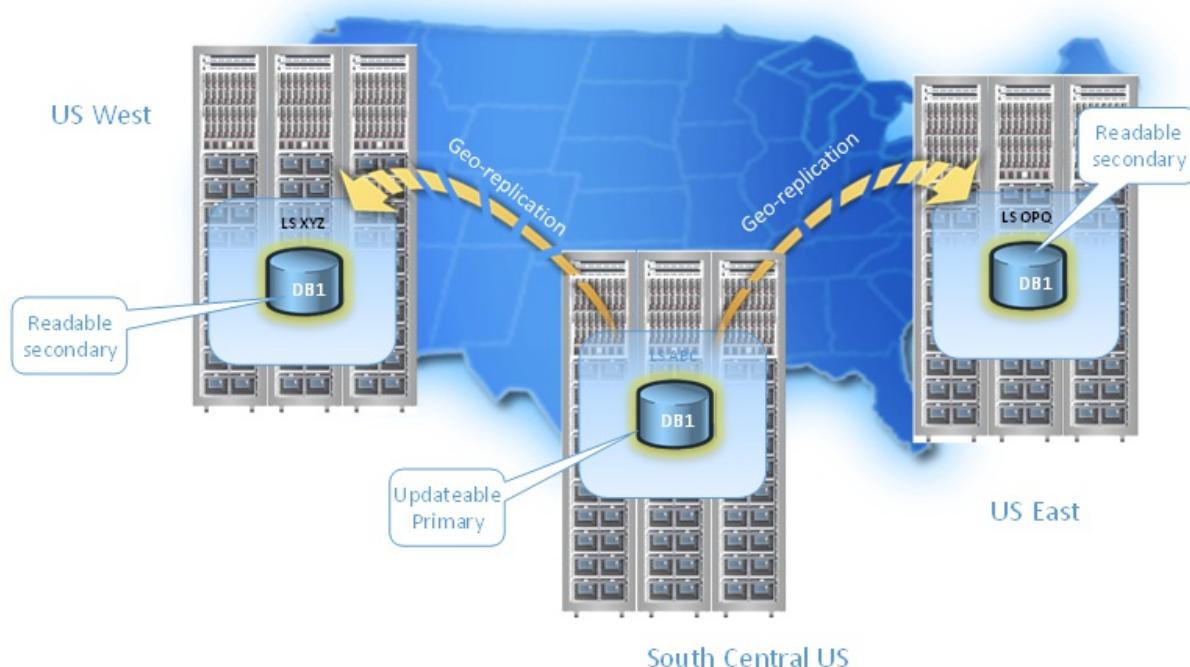
## Next steps

- To learn about service-generated automatic backups, see [automatic backups](#)
- To learn about long-term backup retention, see [long-term backup retention](#)
- To learn about restoring from backups, see [restore from backup](#)

# Overview: Active geo-replication and auto-failover groups

10/29/2018 • 23 minutes to read • [Edit Online](#)

Active geo-replication is Azure SQL Database feature that allows you to create readable replicas of your database in the same or different data center (region).



Active geo-replication is designed as a business continuity solution that allows the application to perform quick disaster recovery in case of a data center 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.

## NOTE

Active geo-replication is available for all databases in all service tiers in all regions. Active Geo-replication is not available in Managed Instance.

Auto-failover groups is an extension of active geo-replication. It is designed to manage the failover of multiple geo-replicated databases simultaneously using an application initiated failover or by delegating failover to be done by the SQL Database service based on a user defined criteria. The latter 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. 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. Both primary and secondary servers for the databases in the failover group must be in the same subscription. Auto-failover groups support replication of all databases in the group to only one secondary server in a different region.

**NOTE**

Use active geo-replication if multiple secondaries are required.

If you are using active geo-replication and 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. If you are using auto-failover groups to manage database recovery and any outage that impacts one or several of the databases in the group results in automatic failover. You can configure the auto-failover policy that best meets your application needs, or you can opt out and use manual activation. 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 end-points to the new region.

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](#)
- [PowerShell: Failover Group](#)
- [Transact-SQL: Single database or elastic pool](#)
- [REST API: Single database](#)
- [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](#). This applies both to active geo-replication and auto-failover groups.

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](#).

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.

**Geo-Replication**  
auditing-server/dimadhus\_d111  
Select a region on the map or from the Target Regions list to create a secondary database.

| PRIMARY                             |                  |                               |
|-------------------------------------|------------------|-------------------------------|
| <input checked="" type="checkbox"/> | North Central US | auditing-server/dimadhus_d111 |
|                                     | Online           |                               |

| SECONDARIES                         |                  |                          |
|-------------------------------------|------------------|--------------------------|
| <input checked="" type="checkbox"/> | 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 capabilities

The active geo-replication feature provides the following essential 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.

- **Multiple readable secondaries**

Two or more secondary databases increase redundancy and level of protection for the primary database and application. If multiple secondary databases exist, the application remains protected even if one of the secondary databases fails. If there is only one secondary database, and it fails, the application is exposed to higher risk until a new secondary database is created.

**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.

- **Support of elastic pool databases**

Each replica can separately participate in an Elastic Pool or not be in any elastic pool at all. The pool choice for each replica is separate and does not depend upon the configuration of any other replica (whether Primary or Secondary). Each Elastic Pool is contained within a single region, therefore multiple replicas in the same topology can never share an Elastic Pool.

- **Configurable compute size of the 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. A secondary with lower compute size is at risk of an increased replication lag, potential unavailability of the secondary, and consequently at risk of substantial data loss after a failover. As a result, the published RPO = 5 sec cannot be guaranteed. The other risk is that after failover the application's performance will be impacted due to insufficient compute capacity of the new primary until it is upgraded to a higher compute size. The time of the upgrade depends on the database size. In addition, currently such upgrade requires that both primary and secondary databases are online and, therefore, cannot be completed until the outage is mitigated. 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. For more information on the SQL Database compute sizes, see [What are SQL Database Service Tiers](#).

- **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.

- **Keeping credentials and firewall rules in sync**

We recommend using [database firewall rules](#) for geo-replicated databases so these rules can be replicated with the database to ensure all secondary databases have the same 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.

## Auto-failover group capabilities

Auto-failover groups feature provides a powerful abstraction of active geo-replication by supporting group level replication and automatic failover. In addition, it removes the necessity to change the SQL connection string after failover by providing the additional listener end-points.

**NOTE**

Auto-failover is not available in Managed Instance.

- **Failover group**

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.

- **Primary server**

A server that hosts the primary databases in the failover group.

- **Secondary server**

A server that hosts the secondary databases in the failover group. The secondary server cannot be in

the same region as the primary server.

- **Adding databases to failover group**

You can put several databases within a server or within an elastic pool into the same failover group. If you add a single database to the group, it automatically creates a secondary database using the same edition and compute size. If the primary database is in an elastic pool, the secondary is automatically created in the elastic pool with the same name. If you add a database that already has a secondary database in the secondary server, that geo-replication is inherited by the group.

**NOTE**

When adding 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.

- **Failover group read-write listener**

A DNS CNAME record formed as **<failover-group-name>.database.windows.net** that points to the current primary server URL. It allows the read-write SQL applications to transparently reconnect to the primary database when the primary changes after failover.

- **Failover group read-only listener**

A DNS CNAME record formed as **<failover-group-name>.secondary.database.windows.net** that points to the secondary server's URL. It allows the read-only SQL applications to transparently connect to the secondary database using the specified load-balancing rules.

- **Automatic failover policy**

By default, the failover group is configured with an automatic failover policy. The system 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.

- **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 readonly 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. In that case the read-only traffic will be automatically redirected to the primary server if the secondary server is not available.

- **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. You can initiate forced or friendly failover (with full data synchronization). The latter could be used to relocate the active server to the primary region. When failover is completed, the DNS records are automatically updated to ensure connectivity to the correct server.

- **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.

#### NOTE

When system detects that the databases in the group are still online (for example, the outage only impacted the service control plane), it immediately activates the failover with full data synchronization (friendly failover) regardless of the value set by **GracePeriodWithDataLossHours**. This behavior ensures that there is no data loss during the recovery. The grace period takes effect only when a friendly failover is not possible. If the outage is mitigated before the grace period expires, the failover is not activated.

- **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.

## Best practices of using failover groups for business continuity

When designing a service with business continuity in mind, you should follow these guidelines:

- **Use 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.

- **Use read-write listener for OLTP workload**

When performing OLTP operations, use **<failover-group-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.

- **Use 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 **<failover-group-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**.

- **Be prepared for perf degradation**

SQL failover decision is independent from the rest of the application or other services used. The application may be “mixed” with some components in one region and some in another. To avoid the degradation, ensure the redundant application deployment in the DR region and follow the guidelines in this article. Note the application in the DR region does not have to use a different connection string.

- **Prepare 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.

#### 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.

## 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. You should 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.

## Upgrading or downgrading a primary database

You can upgrade or downgrade a primary database to a different compute size (within the same service tier) 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.

## 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.

## Programmatically managing failover groups and active geo-replication

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](#).

## Manage SQL database failover using Transact-SQL

| COMMAND   | DESCRIPTION   |
|---|---|
| <a href="#">ALTER DATABASE (Azure SQL Database)</a>                     | Use ADD SECONDARY ON SERVER argument to create a secondary database for an existing database and starts data replication          |
| <a href="#">ALTER DATABASE (Azure SQL Database)</a>                     | Use FAILOVER or FORCE_FAILOVER_ALLOW_DATA_LOSS to switch a secondary database to be primary to initiate failover                  |
| <a href="#">ALTER DATABASE (Azure SQL Database)</a>                     | Use REMOVE SECONDARY ON SERVER to terminate a data replication between a SQL Database and the specified secondary database.       |
| <a href="#">sys.geo_replication_links (Azure SQL Database)</a>          | Returns information about all existing replication links for each database on the Azure SQL Database logical server.              |
| <a href="#">sys.dm_geo_replication_link_status (Azure SQL Database)</a> | Gets the last replication time, last replication lag, and other information about the replication link for a given SQL database.  |
| <a href="#">sys.dm_operation_status (Azure SQL Database)</a>            | Shows the status for all database operations including the status of the replication links.                                       |
| <a href="#">sp_wait_for_database_copy_sync (Azure SQL Database)</a>     | causes the application to wait until all committed transactions are replicated and acknowledged by the active secondary database. |

## Manage SQL database failover using PowerShell

| CMDLET   | DESCRIPTION   |
|--|---|
| <a href="#">Get-AzureRmSqlDatabase</a>                 | Gets one or more databases.   |
| <a href="#">New-AzureRmSqlDatabaseSecondary</a>        | Creates a secondary database for an existing database and starts data replication.                |
| <a href="#">Set-AzureRmSqlDatabaseSecondary</a>        | Switches a secondary database to be primary to initiate failover.                                 |
| <a href="#">Remove-AzureRmSqlDatabaseSecondary</a>     | Terminates data replication between a SQL Database and the specified secondary database.          |
| <a href="#">Get-AzureRmSqlDatabaseReplicationLink</a>  | Gets the geo-replication links between an Azure SQL Database and a resource group or SQL Server.  |
| <a href="#">New-AzureRmSqlDatabaseFailoverGroup</a>    | This command creates a failover group and registers it on both primary and secondary servers      |
| <a href="#">Remove-AzureRmSqlDatabaseFailoverGroup</a> | Removes the failover group from the server and deletes all secondary databases included the group |
| <a href="#">Get-AzureRmSqlDatabaseFailoverGroup</a>    | Retrieves the failover group configuration  |

| CMDLET   | DESCRIPTION   |
|--|---|
| <a href="#">Set-AzureRmSqlDatabaseFailoverGroup</a>    | Modifies the configuration of the failover group                |
| <a href="#">Switch-AzureRMSqlDatabaseFailoverGroup</a> | Triggers failover of the failover group to the secondary server |

### IMPORTANT

For sample scripts, see [Configure and failover a single database using active geo-replication](#), [Configure and failover a pooled database using active geo-replication](#), and [Configure and failover a failover group for a single database](#).

## Manage SQL database failover using the REST API

| API  | DESCRIPTION   |
|--|---|
| <a href="#">Create or Update Database (createMode=Restore)</a>         | Creates, updates, or restores a primary or a secondary database.  |
| <a href="#">Get Create or Update Database Status</a>                   | Returns the status during a create operation.   |
| <a href="#">Set Secondary Database as Primary (Planned Failover)</a>   | Sets which replica database is primary by failing over from the current primary replica database.   |
| <a href="#">Set Secondary Database as Primary (Unplanned Failover)</a> | Sets which replica database is primary by failing over from the current primary replica database. This operation might result in data loss.                                     |
| <a href="#">Get Replication Link</a>                                   | 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. |
| <a href="#">Replication Links - List By Database</a>                   | 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.       |
| <a href="#">Delete Replication Link</a>                                | Deletes a database replication link. Cannot be done during failover.  |
| <a href="#">Create or Update Failover Group</a>                        | Creates or updates a failover group   |
| <a href="#">Delete Failover Group</a>                                  | Removes the failover group from the server  |
| <a href="#">Failover (Planned)</a>                                     | Fails over from the current primary server to this server.  |
| <a href="#">Force Failover Allow Data Loss</a>                         | Fails over from the current primary server to this server. This operation might result in data loss.  |
| <a href="#">Get Failover Group</a>                                     | Gets a failover group.  |
| <a href="#">List Failover Groups By Server</a>                         | Lists the failover groups in a server.  |
| <a href="#">Update Failover Group</a>                                  | Updates a failover group.   |

| API | DESCRIPTION |
|-----|-------------|
|     |             |

## 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](#)
  - [Configure and failover a failover group for a single database](#)
- 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 active geo-replication for Azure SQL Database in the Azure portal and initiate failover

9/25/2018 • 3 minutes to read • [Edit Online](#)

This article shows you how to configure active geo-replication for SQL Database in the [Azure portal](#) and to initiate failover.

To initiate failover with the Azure portal, see [Initiate a planned or unplanned failover for Azure SQL Database with the Azure portal](#).

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

Active geo-replication must be between databases in the same subscription.

## Add a secondary database

The following steps create a new secondary database in a geo-replication partnership.

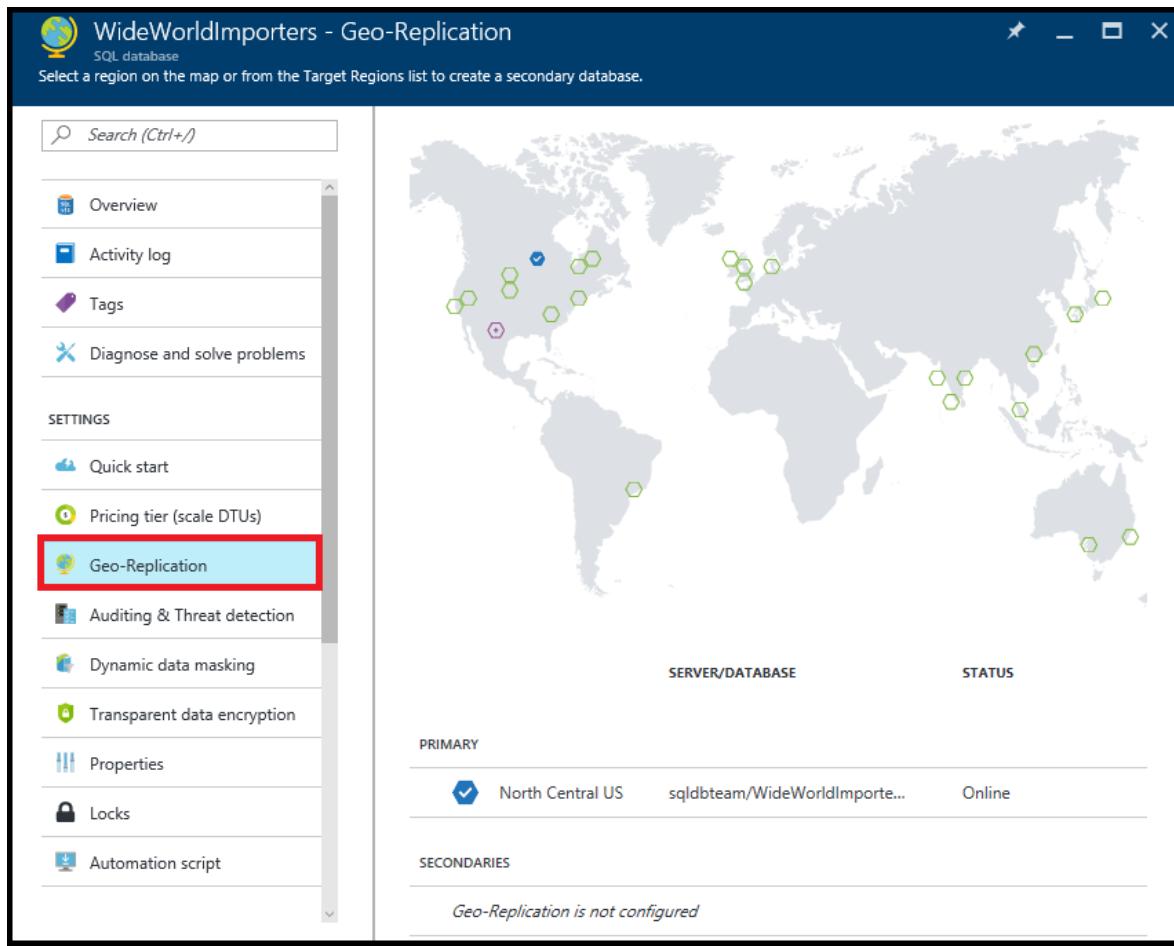
To add a secondary database, you must be the subscription owner or co-owner.

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 database in an elastic pool. 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.

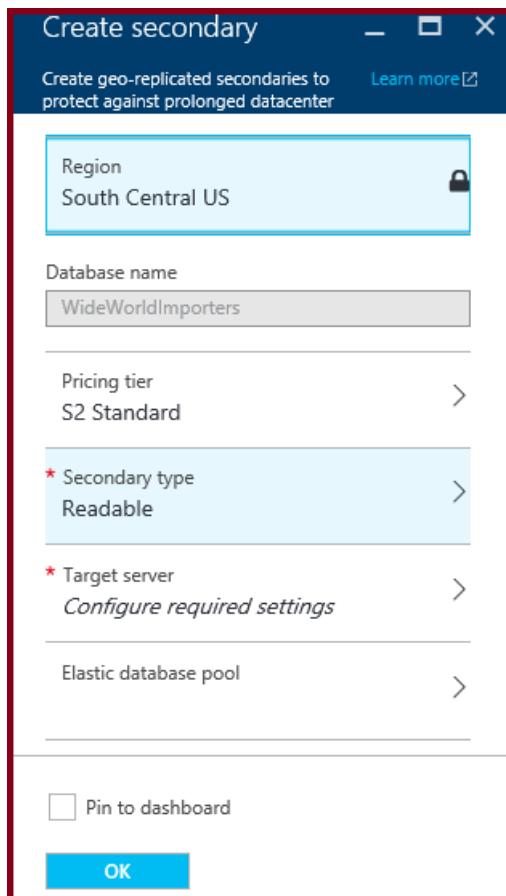
## NOTE

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](#).



3. Select or configure the server and pricing tier for the secondary database.



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.

The screenshot shows a world map with various green hexagonal icons representing geo-replication partners. A specific location in North America is highlighted with a dashed blue border. Below the map is a table with two sections: 'PRIMARY' and 'SECONDARIES'. The 'PRIMARY' section shows one entry: 'North Central US' with status 'Online'. The 'SECONDARIES' section shows one entry: 'South Central US' with status 'Initializing...'. The 'SECONDARIES' section is highlighted with a red rectangular box.

| SERVER/DATABASE  | STATUS                       |
|------------------|------------------------------|
| PRIMARY          |                              |
| North Central US | sqldbteam/WideWorldImport... |
| Online           |                              |
| SECONDARIES      |                              |
| South Central US | server101801/WideWorldImp... |
|                  | Initializing...              |

7. When the seeding process is complete, the secondary database displays its status.

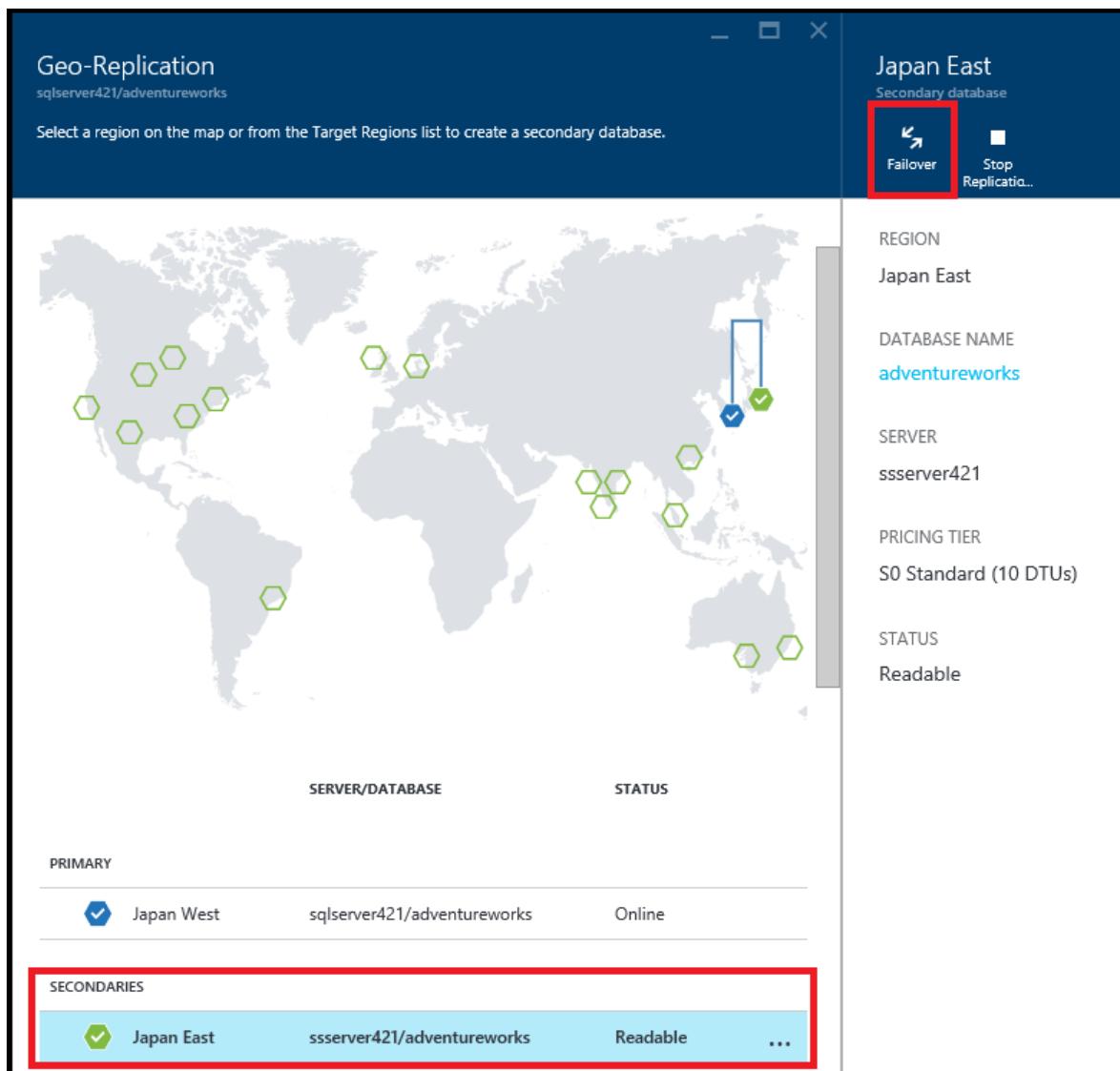
The screenshot shows the same table structure as the previous screenshot, but the 'SECONDARIES' section has been updated. The 'South Central US' entry now has a green checkmark icon and the status 'Readable'. The 'Readable' status is highlighted with a red rectangular box.

| SERVER/DATABASE  | STATUS                       |
|------------------|------------------------------|
| PRIMARY          |                              |
| North Central US | sqldbteam/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**.



4. Click **Yes** to begin the failover.

The command immediately switches the secondary database into the primary role.

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

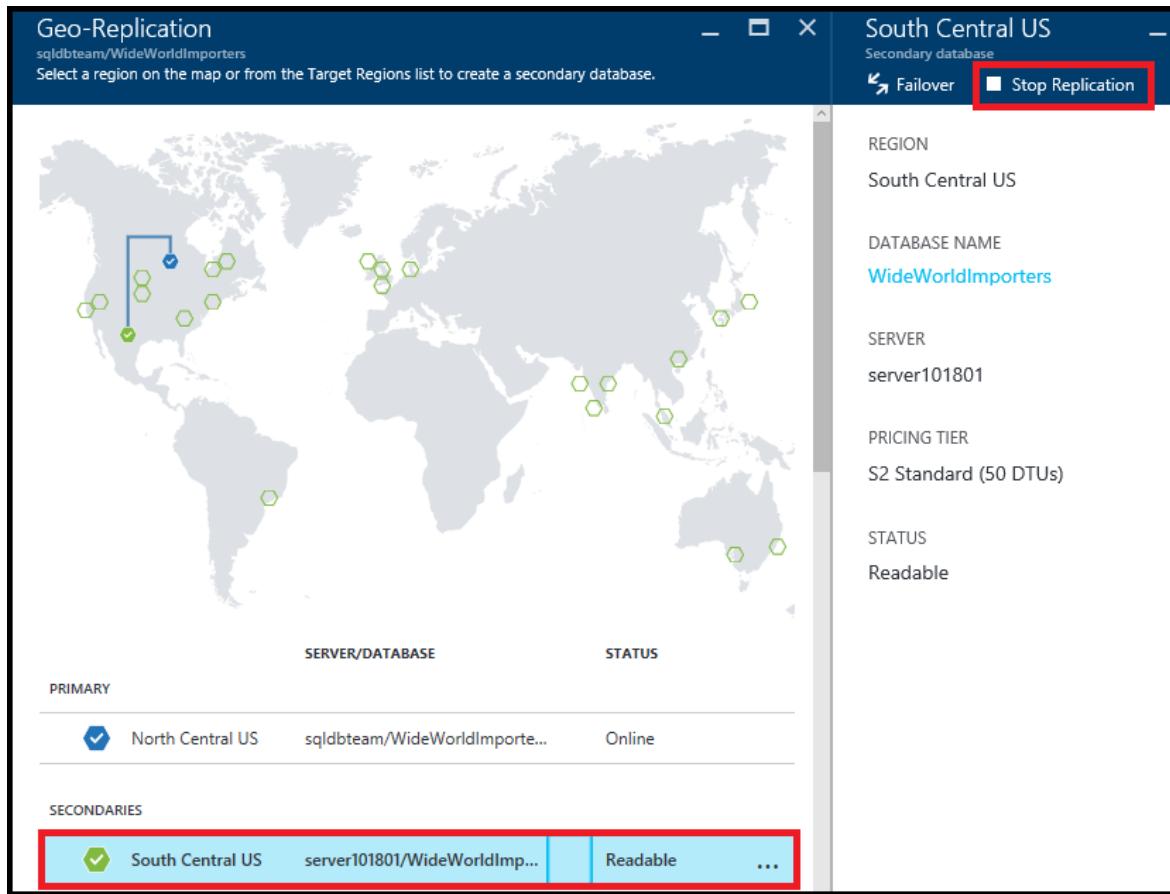
This command is designed for quick recovery of the database in case of an outage. It triggers failover without data synchronization (forced failover). If the primary is online and committing transactions when the command is issued some data loss may occur.

## 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**.



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](#).
- For a business continuity overview and scenarios, see [Business continuity overview](#).

# Configure and manage Azure SQL Database security for geo-restore or failover

9/25/2018 • 4 minutes to read • [Edit Online](#)

This topic describes the authentication requirements to configure and control [active geo-replication](#) and the steps required to set up user access to the secondary database. 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](#).

## NOTE

[Active geo-replication](#) is now available for all databases in all service tiers.

## 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.

### 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 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](#).
- For information about using and configuring active geo-replication, see [active geo-replication](#)
- For information about using geo-restore, see [geo-restore](#)

# Recover an Azure SQL database using automated database backups

10/24/2018 • 8 minutes to read • [Edit Online](#)

By default, SQL Database backups are stored in geo-replicated blob storage (RA-GRS). The following options are available for database recovery using [automated database backups](#):

- Create a new database on the same logical server recovered to a specified point in time within the retention period.
- Create a database on the same logical server recovered to the deletion time for a deleted database.
- Create a new database on any logical server in the same region recovered to the point of the most recent backups.
- Create a new database on any logical 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 LTR backup on any logical server in any region.

## IMPORTANT

You cannot overwrite an existing database during restore.

When using Standard or Premium service tier, a restored database incurs an extra storage cost under the following conditions:

- Restore of P11–P15 to S4-S12 or P1–P6 if the database max size is greater than 500 GB.
- Restore of P1–P6 to S4-S12 if the database max size is greater than 250 GB.

The extra cost is because the max size of the restored database is greater than the amount of storage included for the compute size, and any extra storage provisioned above the included amount is charged extra. For pricing details of extra storage, see the [SQL Database pricing page](#). If the actual amount of space used is less than the amount of storage included, then this extra cost can be avoided by reducing the database max size to the included amount.

## NOTE

[Automated database backups](#) are used when you create a [database copy](#).

## Recovery time

The recovery time to restore a database using automated database backups is impacted 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 very large and/or active database, the restore may take several hours. If there is prolonged outage in a region, it is possible that there are large numbers of geo-restore requests being processed by other regions. When there are many requests, the recovery time may increase for databases in that region. Most database restores complete in less than 12 hours.

For a single subscription, there are some limitations on number of concurrent restore requests (including point in time restore, geo restore and restore from long-term retention backup) being submitted and proceeded:

|                                    | 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 is no built-in functionality to do bulk restore. The [Azure SQL Database: Full Server Recovery](#) script is an example of one way of accomplishing this task.

#### IMPORTANT

To recover using automated backups, you must be a member of the SQL Server Contributor role in the subscription or be the subscription owner - see [RBAC: Built-in roles](#). You can recover using the Azure portal, PowerShell, or the REST API. You cannot use Transact-SQL.

## Point-in-time restore

You can restore a single, pooled, or Managed Instance database to an earlier point in time as a new database on the same server using the Azure portal, [PowerShell](#), or the [REST API](#). A database can be restored to any service tier or compute size. Ensure you have sufficient resources on the server to which you are restoring the database. Once complete, the restored database is a normal, fully accessible, online database. The restored database is charged at normal rates based on its service tier and compute size. You do not incur charges until the database restore is complete.

You generally restore a database to an earlier point for recovery purposes. When doing so, you can treat the restored database as a replacement for the original database or use it to retrieve data from and then update the original database.

- **Database replacement**

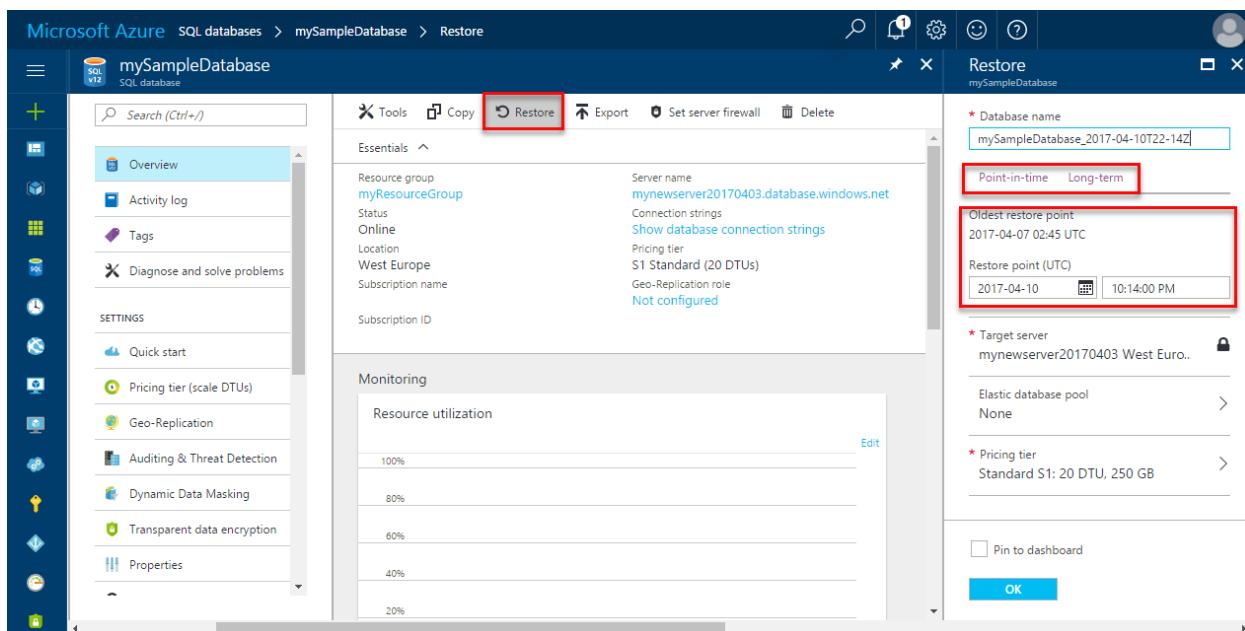
If the restored database is intended as a replacement for the original database, you should verify the compute size and/or service tier are appropriate and scale the database if necessary. You can rename the original database and then give the restored database the original name using the [ALTER DATABASE](#) 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 the necessary data recovery scripts to extract data from the restored database 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 is canceled and you are not charged for the database that did not complete the restore.

To recover a single, pooled, or Managed Instance database to a point in time using the Azure portal, open the

page for your database and click **Restore** on the toolbar.



#### IMPORTANT

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 for a deleted database on the same logical server using the Azure portal, [PowerShell](#), or the [REST \(createMode=Restore\)](#). You can restore a deleted database to an earlier point in time during the retention using [PowerShell](#).

#### NOTE

Restoring deleted database is not available in Managed Instance.

#### TIP

For a sample PowerShell script showing how to restore a deleted database, see [Restore a SQL database using PowerShell](#).

#### IMPORTANT

If you delete an Azure SQL Database server instance, all its databases are also deleted and cannot be recovered. There is currently no support for restoring a deleted server.

## Deleted database restore using the Azure portal

To recover a deleted database using the Azure portal during its [DTU-based model retention period](#) or [vCore-based model retention period](#) using the Azure portal, open the page for your server and in the Operations area, click **Deleted databases**.

Microsoft Azure < mynewserver20170403

mynewserver20170403 SQL server

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

Properties

Locks

Automation script

SUPPORT + TROUBLESHOOTING

Automatic tuning

New support request

New pool Import database Reset password Move Delete

| DATABASE           | STATUS | PRICING TIER |
|--------------------|--------|--------------|
| myMigratedDatabase | Online | Standard: S1 |
| mySampleDatabase   | Online | Standard: S1 |

Elastic database pools

Elastic database pools

0 Elastic database pools

| NAME                   | PRICING TIER | POOL DTU |
|------------------------|--------------|----------|
| No elastic pools found |              |          |

Usage

DTU quota

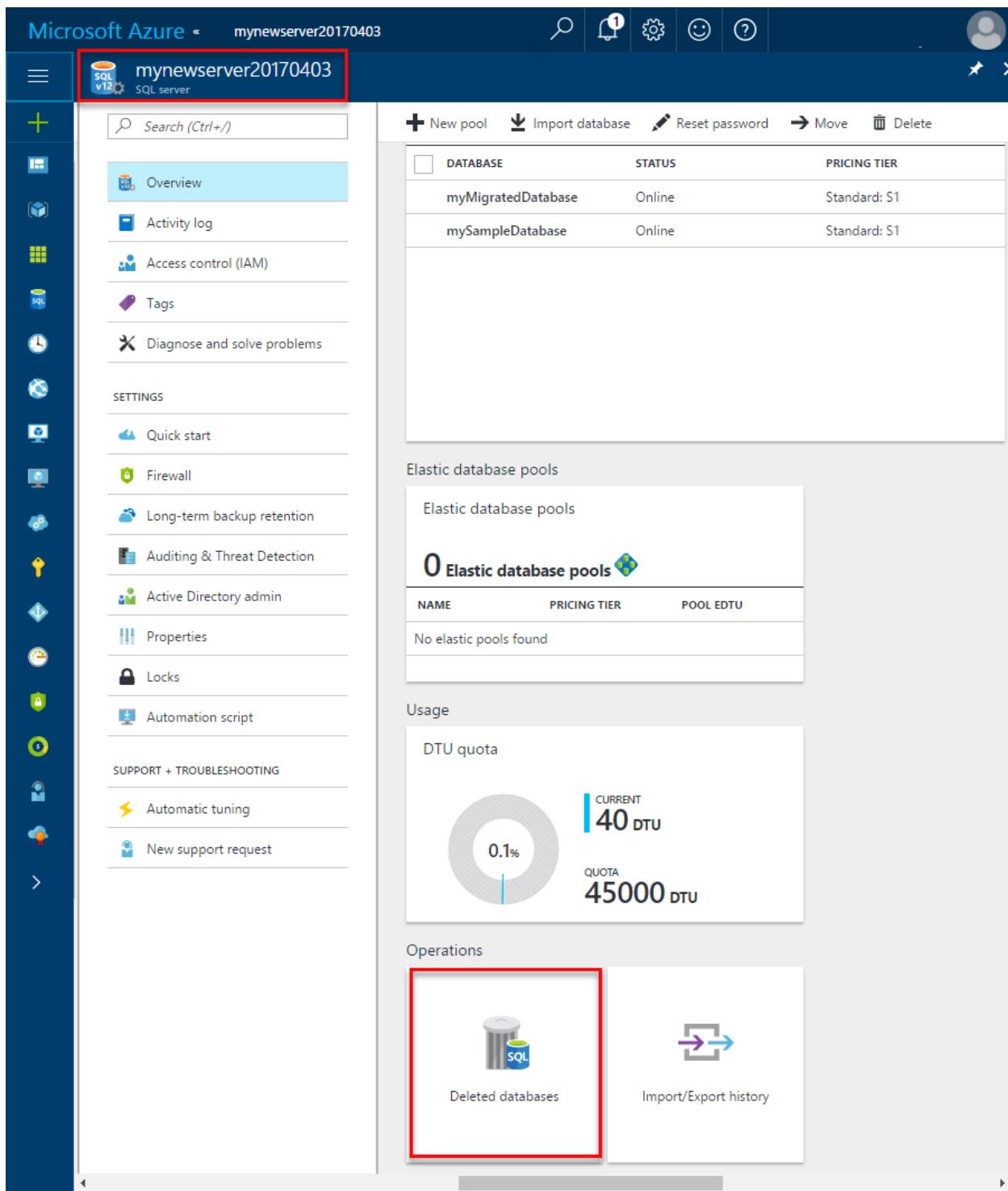
CURRENT 40 DTU

0.1%

QUOTA 45000 DTU

Operations

Deleted databases Import/Export history



Microsoft Azure < mySampleDatabase > mynewserver20170403 > Deleted databases > Restore

Deleted databases mynewserver20170403

| DATABASE         | DELETION TIME (UTC) | CREATION TIME (UTC) | EDITION  |
|------------------|---------------------|---------------------|----------|
| mySampleDatabase | 2017-04-07 02:32    | 2017-04-06 23:36    | Standard |
| mySampleDatabase | 2017-04-06 23:34    | 2017-04-05 21:52    | Standard |

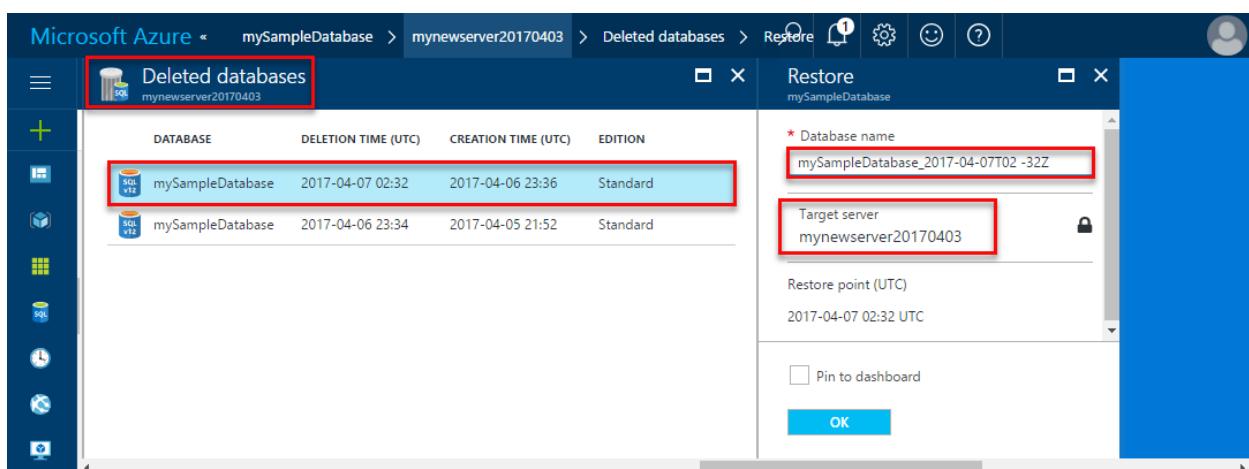
Restore mySampleDatabase

\* Database name: mySampleDatabase\_2017-04-07T02-32Z

Target server: mynewserver20170403

Restore point (UTC): 2017-04-07 02:32 UTC

Pin to dashboard OK



**IMPORTANT**

To programmatically restore a deleted database, see [Programmatically performing recovery using automated backups](#)

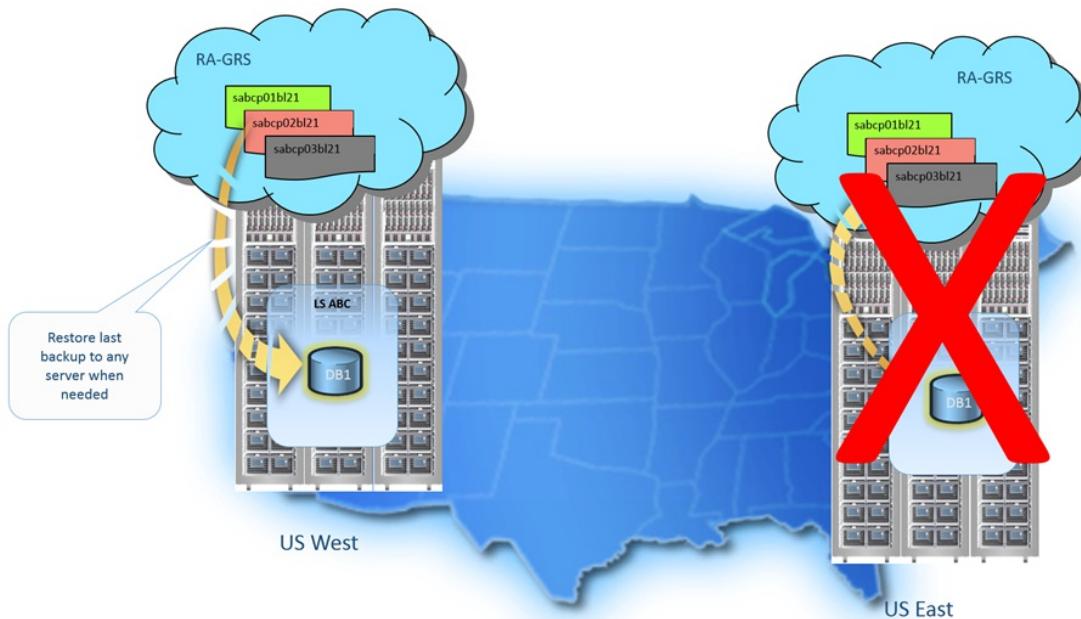
## Geo-restore

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-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.

**NOTE**

Geo-restore is not available in Managed Instance.

Geo-restore is the default recovery option when your database is unavailable because of an incident in the region where the database is hosted. If a large-scale incident in a region results in unavailability of your database application, you can restore a database from the geo-replicated backups 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. This delay can be up to an hour, so, if a disaster occurs, there can be up to one hour data loss. The following illustration shows restore of the database from the last available backup in another region.

**TIP**

For a sample PowerShell script showing how to perform a geo-restore, see [Restore a SQL database using PowerShell](#).

Point-in-time restore on a geo-secondary is not currently supported. Point-in-time restore can be done only on a primary database. For detailed information about using geo-restore to recover from an outage, see [Recover from an outage](#).

## IMPORTANT

Recovery from backups is the most basic of the disaster recovery solutions available in SQL Database with the longest Recovery Point Objective (RPO) and Estimate Recovery Time (ERT). For solutions using small size databases (e.g. Basic service tier or small size tenant databases in elastic pools), geo-restore is frequently a reasonable DR solution with an ERT of up to 12 hours (generally much less). For solutions using large databases and require shorter recovery times, you should consider using [Failover groups and active geo-replication](#). Active geo-replication offers a much lower RPO and ERT as it only requires you initiate a failover to a continuously replicated secondary. For more information on business continuity choices, see [Overview of business continuity](#).

## Geo-restore using the Azure portal

To geo-restore a database during its [DTU-based model retention period](#) or vCore-based model retention period using the Azure portal, open the SQL Databases page and then click **Add**. In the **Select source** text box, select **Backup**. Specify the backup from which to perform the recovery in the region and on the server of your choice.

## Programmatically performing recovery using automated backups

As previously discussed, in addition to the Azure portal, database recovery can be performed programmatically using Azure PowerShell or the REST API. The following tables describe the set of commands available.

### PowerShell

- To restore a single or pooled database, see [Restore-AzureRmSqlDatabase](#)

| CMDLET  | DESCRIPTION                                   |
|---|---|
| <a href="#">Get-AzureRmSqlDatabase</a>              | Gets one or more databases.                   |
| <a href="#">Get-AzureRMSqlDeletedDatabaseBackup</a> | Gets a deleted database that you can restore. |
| <a href="#">Get-AzureRmSqlDatabaseGeoBackup</a>     | Gets a geo-redundant backup of a database.    |
| <a href="#">Restore-AzureRmSqlDatabase</a>          | Restores a SQL database.                      |

### TIP

For a sample PowerShell script showing how to perform a point-in-time restore of a database, see [Restore a SQL database using PowerShell](#).

- To restore a Managed Instance database, see [Point-in-time restore of a database on Azure SQL Managed Instance using AzureRm.Sql PowerShell library](#)

### REST API

To restore a single or pooled database using the REST API:

| API  | DESCRIPTION                                   |
|--|---|
| <a href="#">REST (createMode=Recovery)</a>           | Restores a database                           |
| <a href="#">Get Create or Update Database Status</a> | Returns the status during a restore operation |

### Azure CLI

To restore a single or pooled database using Azure CLI, see [az sql db 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

- 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 long-term retention, see [Long-term retention](#).
- To learn about faster recovery options, see [Failover groups and active geo-replication](#).

# Restore an Azure SQL Database or failover to a secondary

9/25/2018 • 5 minutes to read • [Edit Online](#)

Azure SQL Database offers the following capabilities for recovering from an outage:

- [Active geo-replication and 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.

## 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 logical 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 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. Learn how to [configure failover groups](#).

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](#)

## 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

9/25/2018 • 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 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 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 failover groups](#).

# Designing globally available services using Azure SQL Database

9/24/2018 • 11 minutes to read • [Edit Online](#)

When building and deploying cloud services with Azure SQL Database, you use [failover groups and active geo-replication](#) 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:

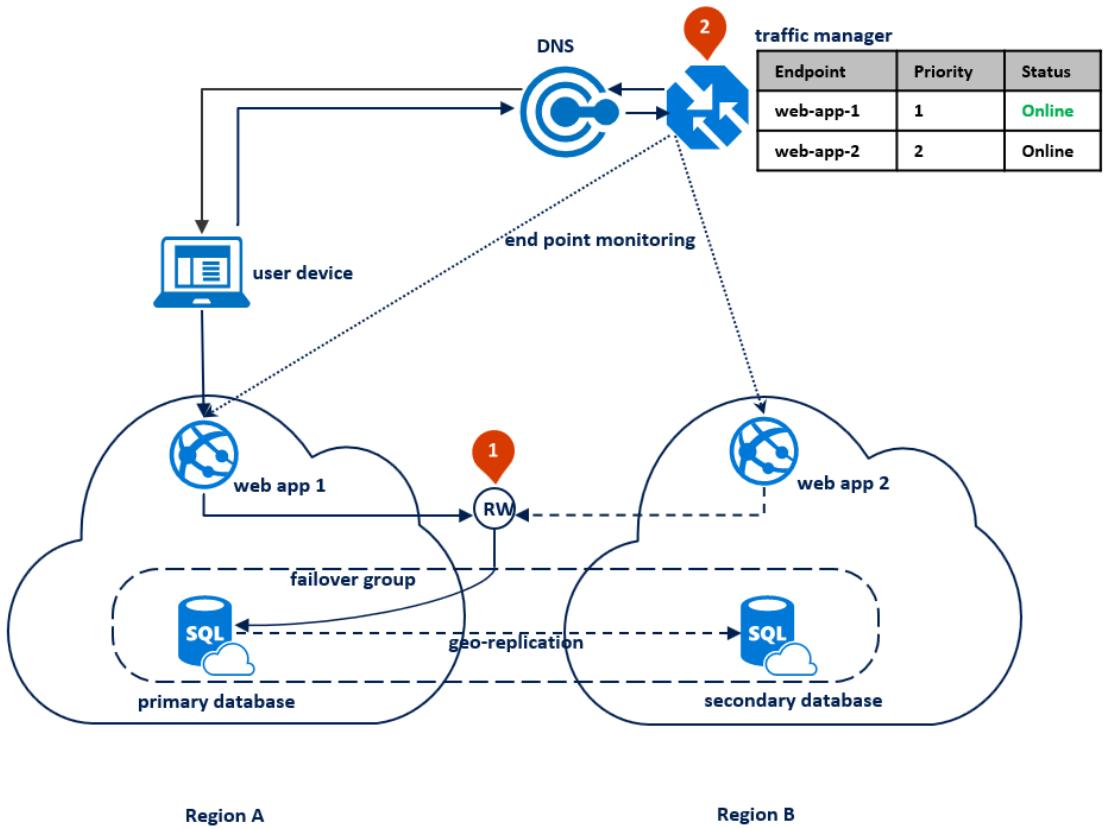
- Application is active in one Azure region
- All database sessions require read and write access (RW) to data
- Web tier and data tier must be collocated to reduce latency and traffic cost
- Fundamentally, downtime is a higher business risk for these applications than data loss

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.

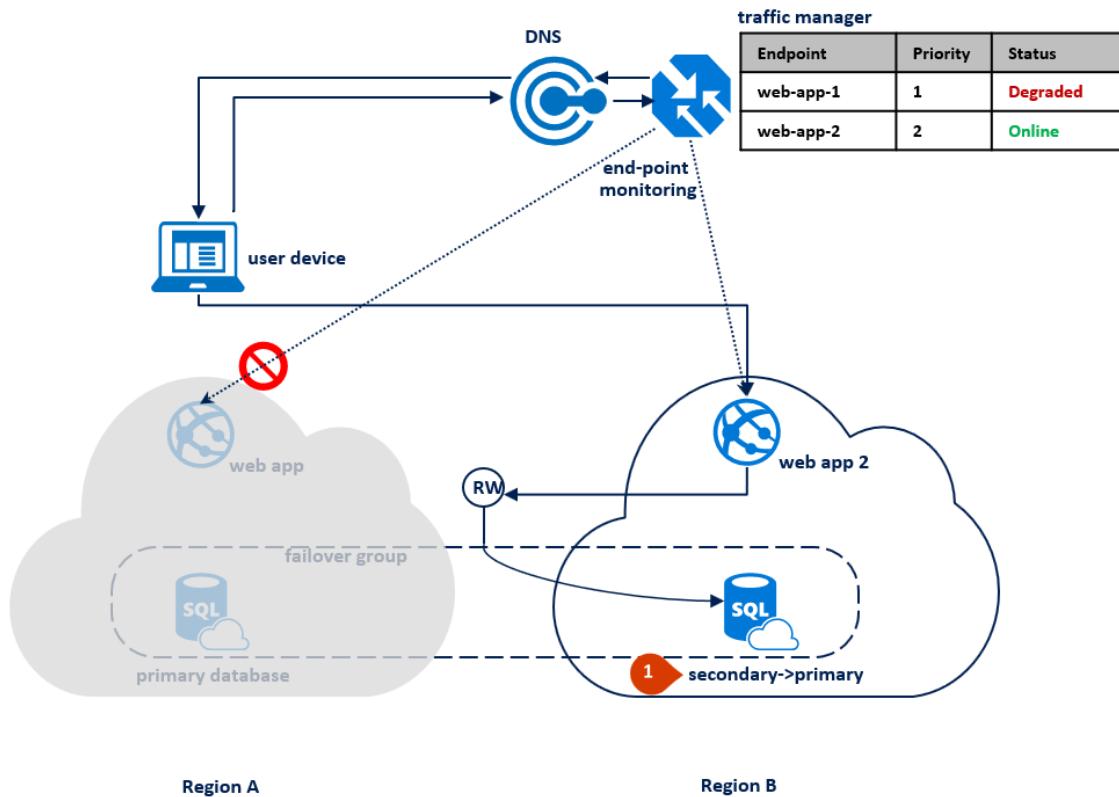
The following diagram shows this configuration before an outage:



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. n>

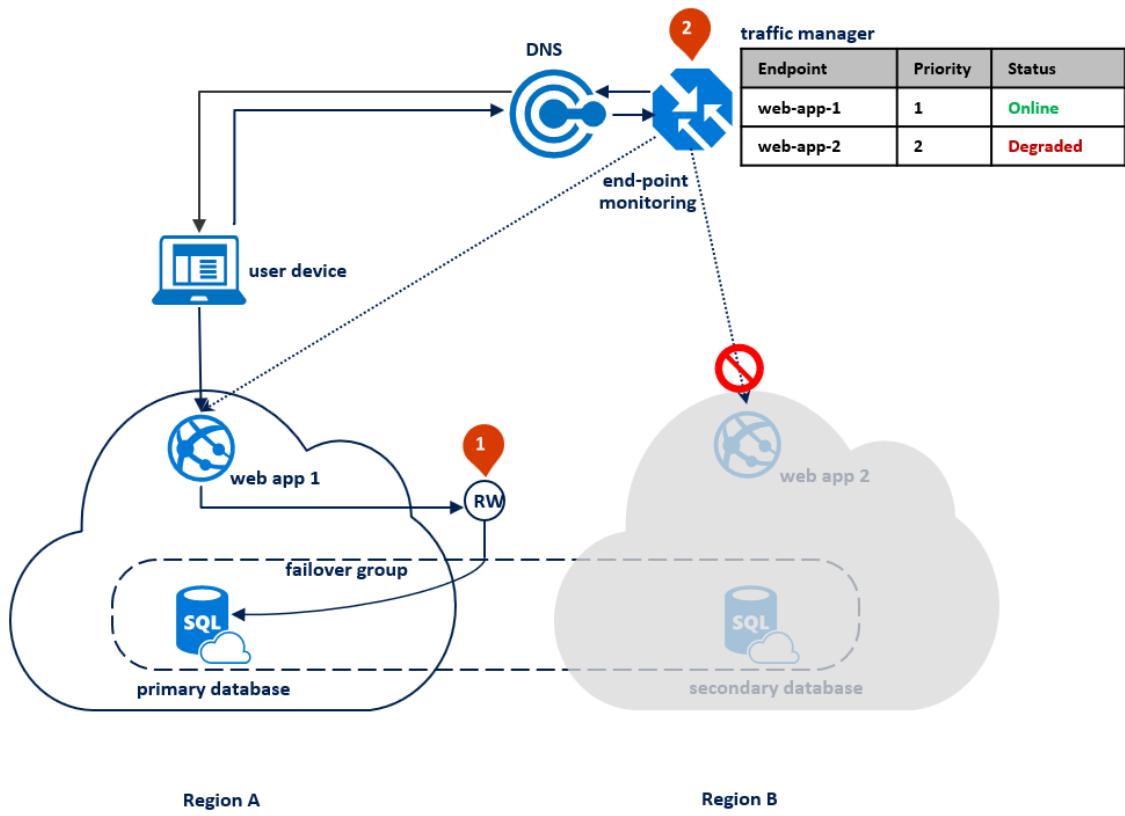


If an outage happens in region B, the replication process between the primary and the secondary database gets suspended but the link between the two remains intact (1). Traffic managed detects that connectivity to Region B is broken and marks the endpoint web app 2 as Degraded (2). The application's 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.

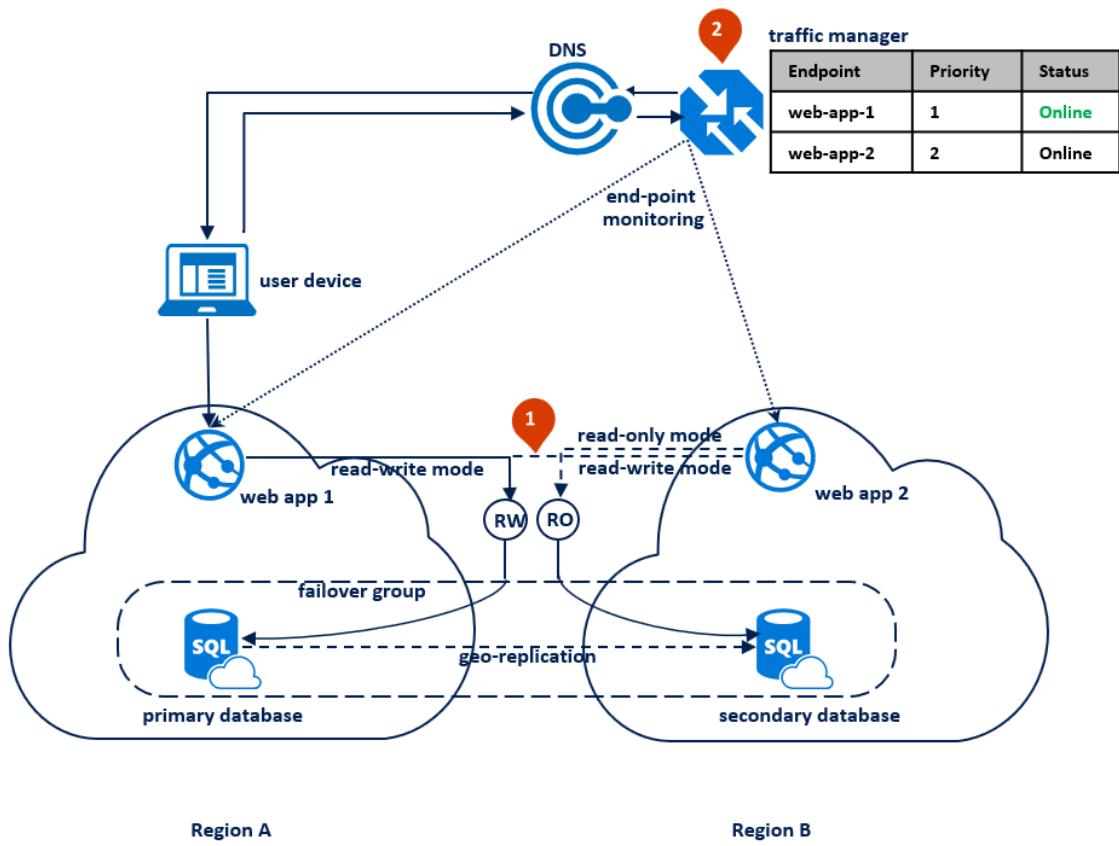
## Scenario 2: Azure regions for business continuity with maximum data preservation

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).

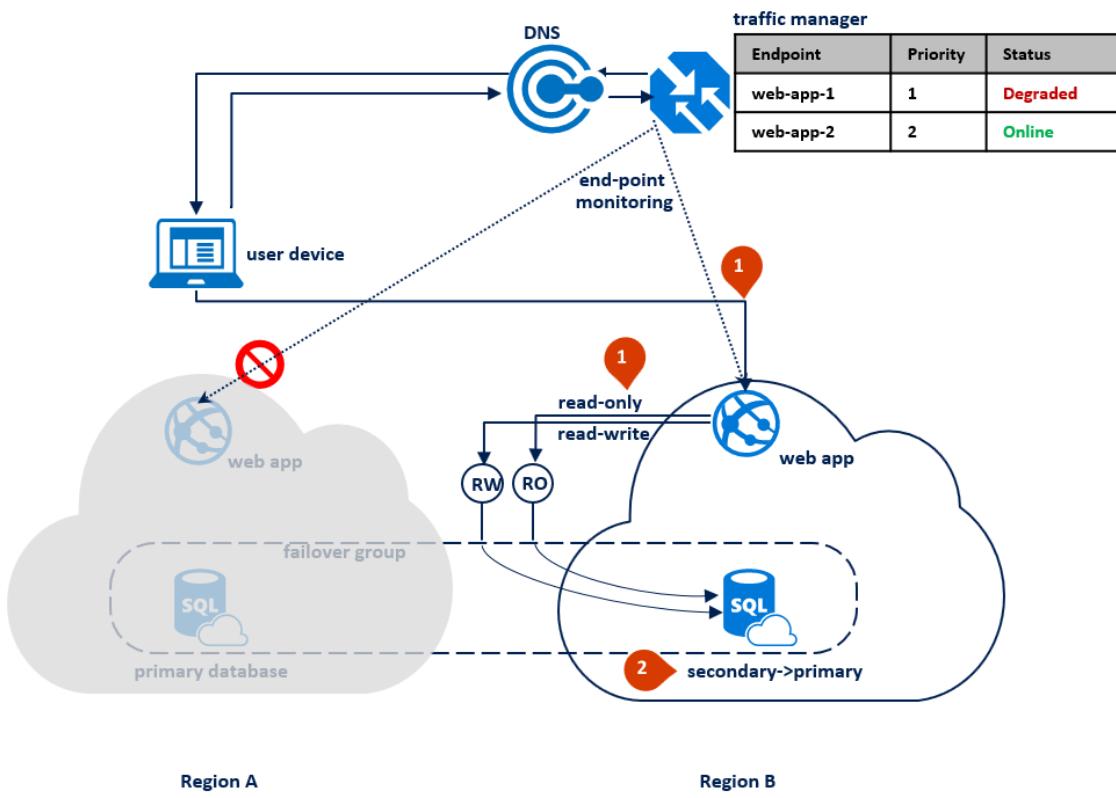
The following diagram illustrates this configuration before an outage:



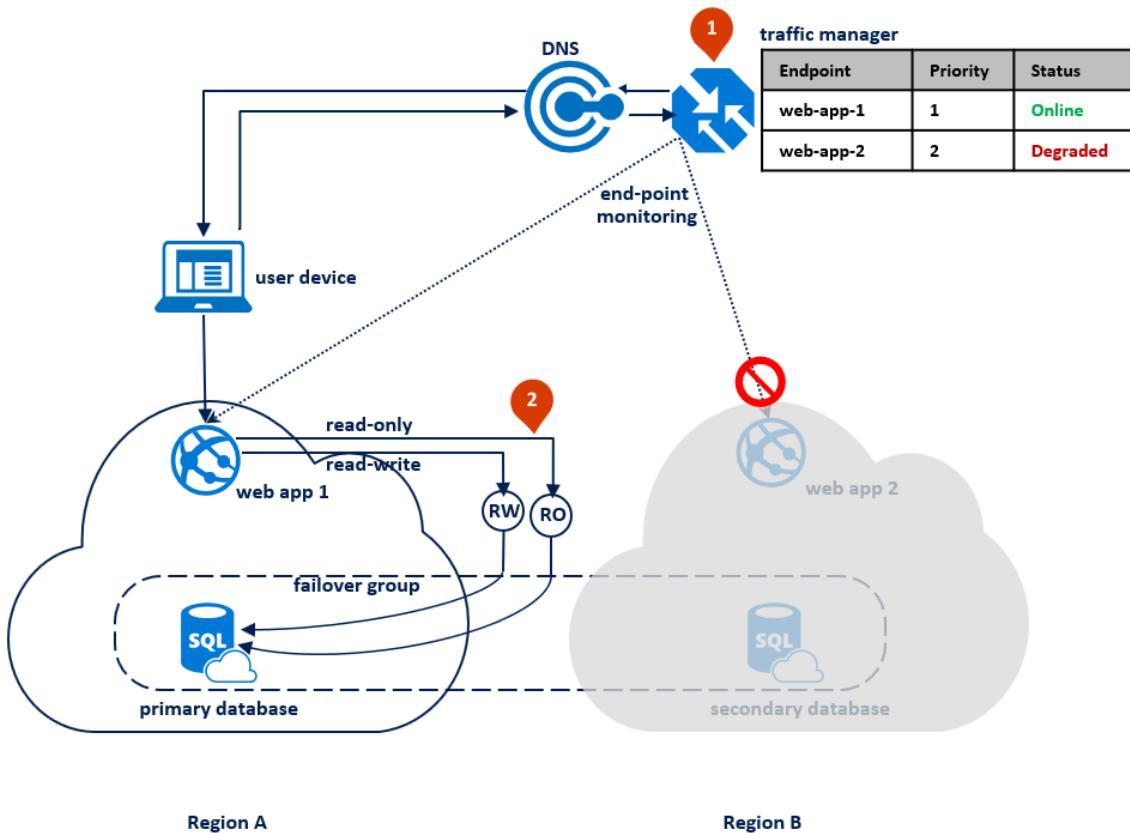
When the traffic manager detects a connectivity failure to region A, it automatically switches user traffic to the application instance in region B. With this pattern, it is important that you set the grace period with data loss to a sufficiently high value, for example 24 hours. It ensures that data 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:

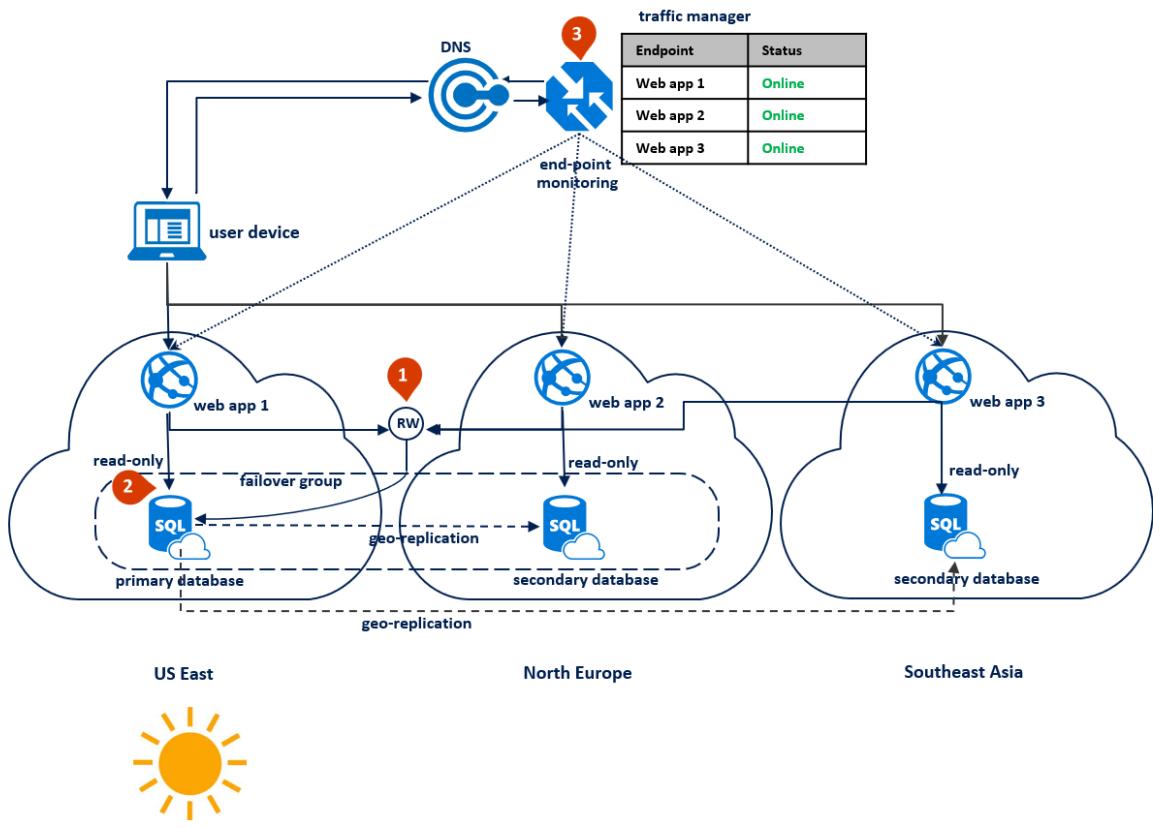
- The end users access the application from different geographies
- The application includes read-only workloads that do not depend on full synchronization with the latest updates
- Write access to data should be supported in the same geography for majority of the users
- Read latency is critical for 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.

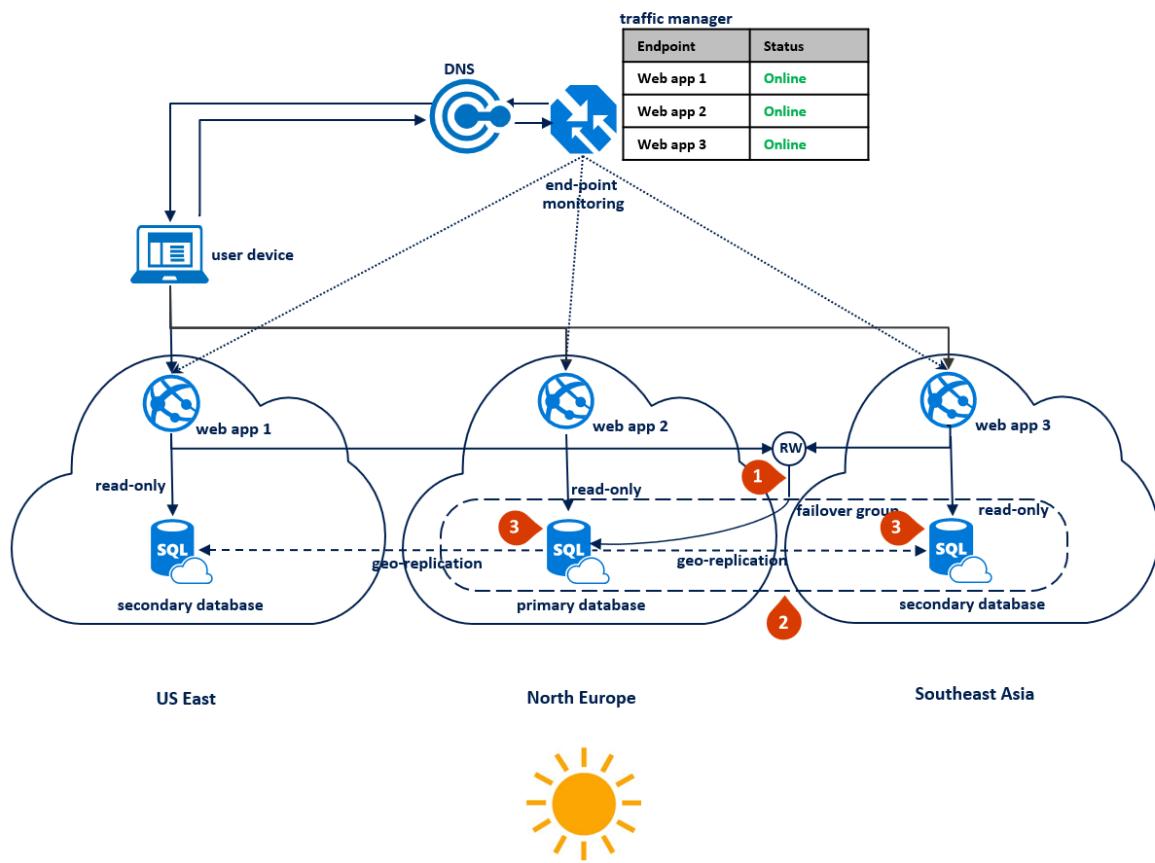


) (oo)

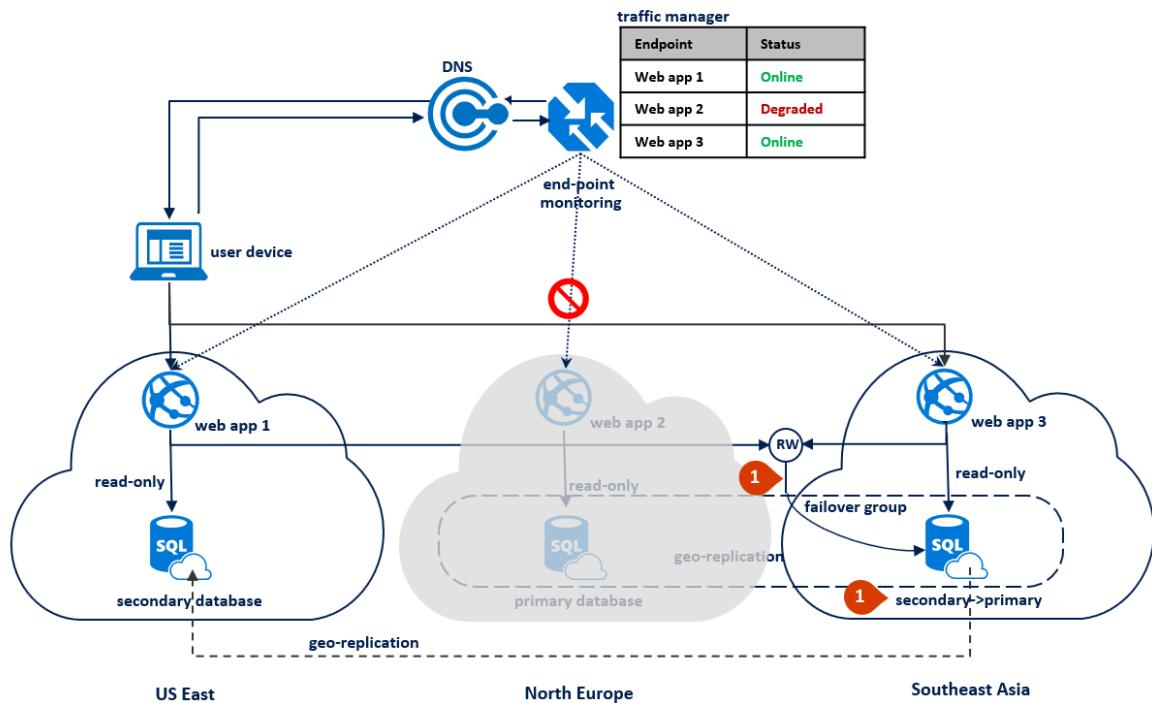
At the end of the day (for example at 11PM local time) the active databases should be switched to the next region (North Europe). This task can be fully automated by using [Azure scheduling service](#). The task involves the following steps:

- 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 scheduler 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 geo-replication and failover groups, see [active geo-replication](#)
- 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

9/25/2018 • 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.

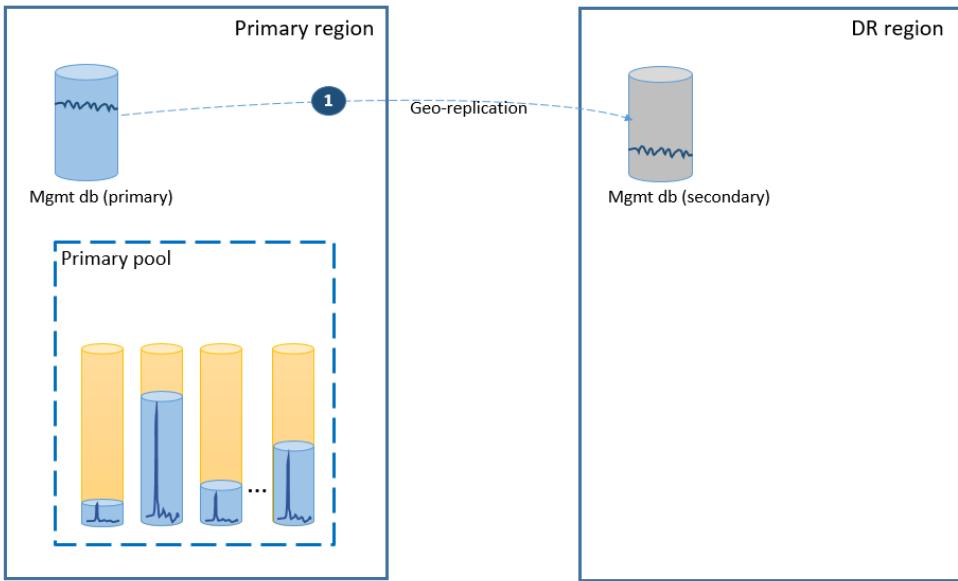
## NOTE

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.*

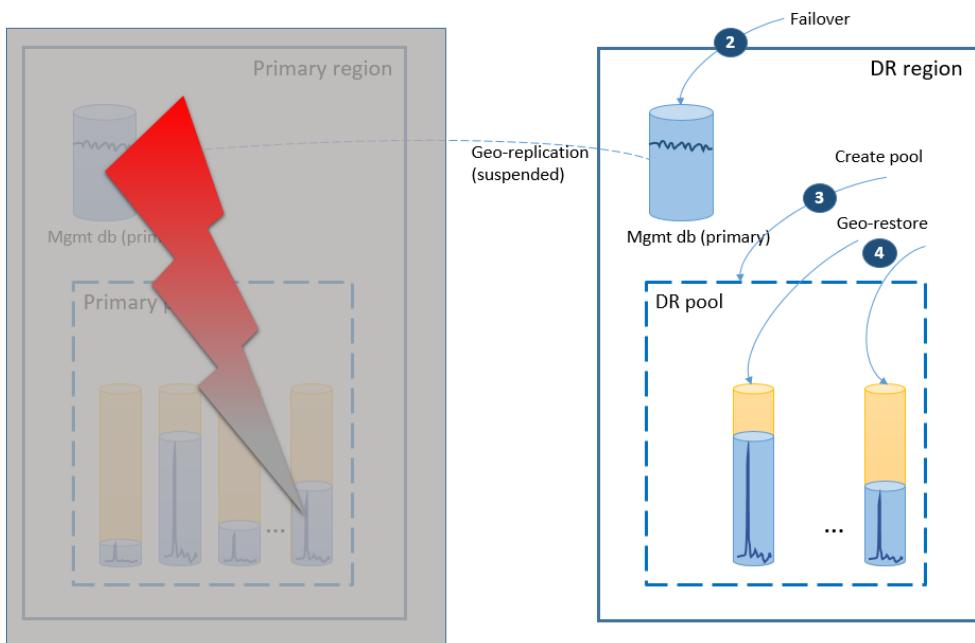
To satisfy the simplicity requirement, deploy all tenant databases into one elastic pool in the Azure region of your choice and deploy management databases as geo-replicated single databases. For the disaster recovery of tenants, use geo-restore, which comes at no additional cost. To ensure the availability 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.

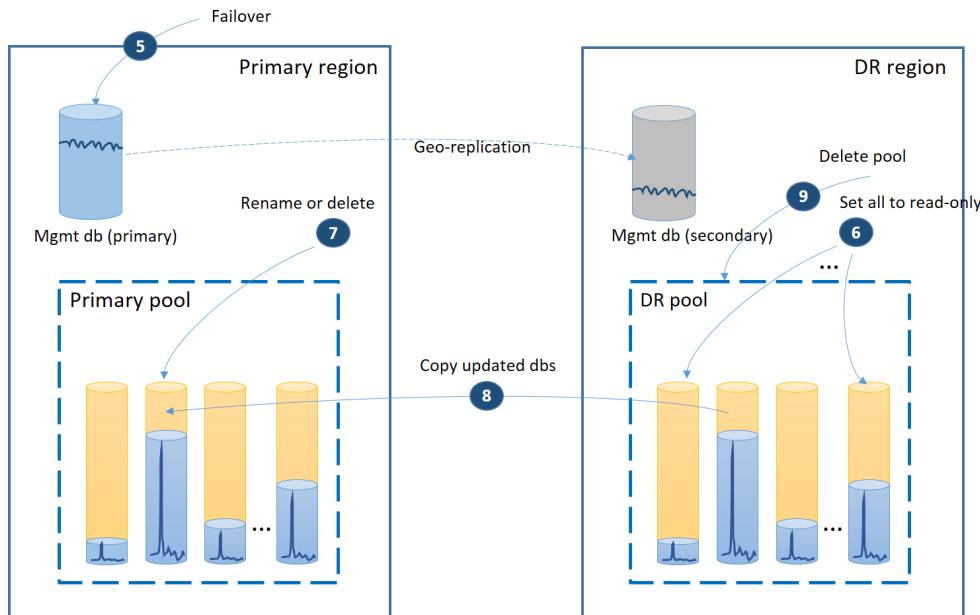
At this point your application is back online in the DR region, but some customers experience delay when accessing their data.



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.



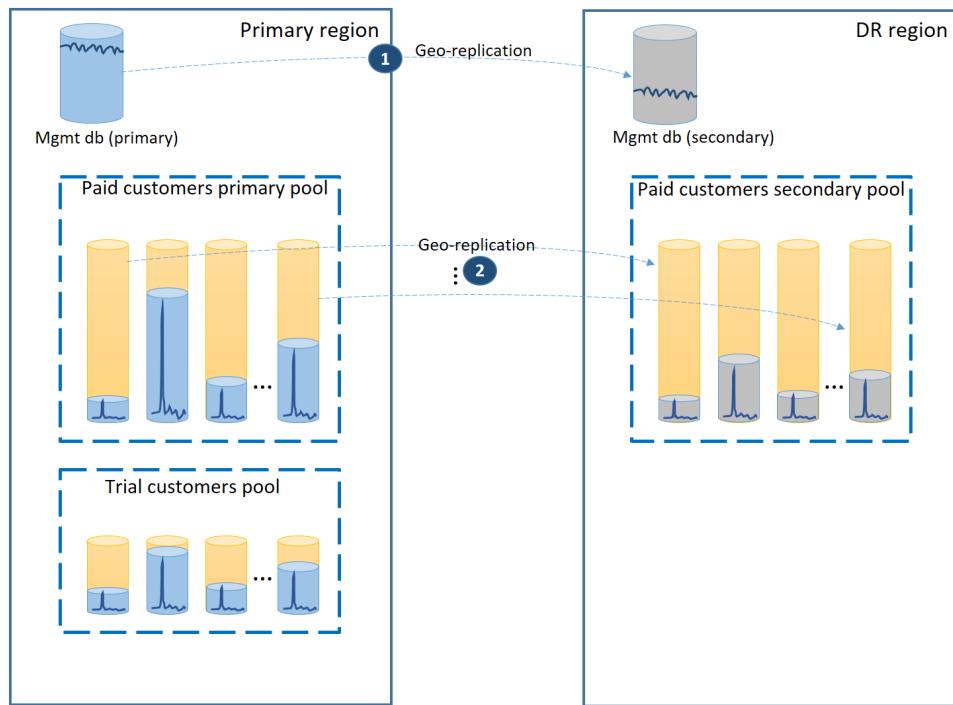
The key **benefit** of this strategy is low ongoing cost for data tier redundancy. Backups are taken automatically by the SQL Database service with no application rewrite and at no additional cost. The cost is incurred only when the elastic databases are restored. The **trade-off** is that the complete 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

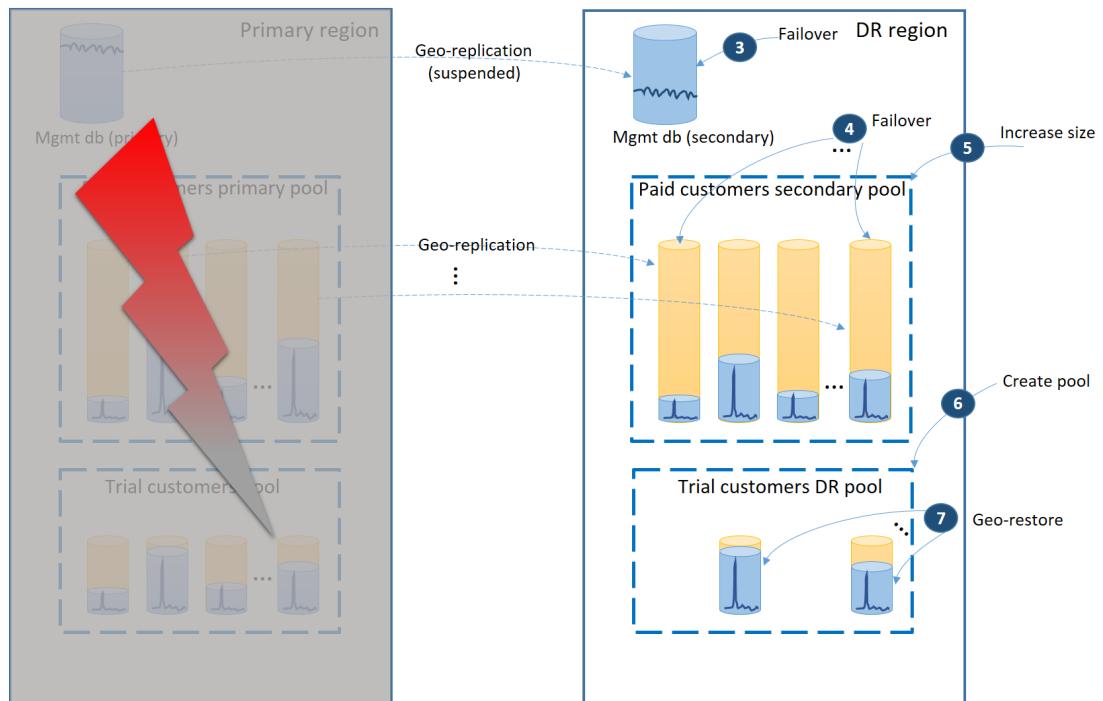
recovery time, the paying customers' tenant databases are geo-replicated. This configuration is illustrated on the next diagram.



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:



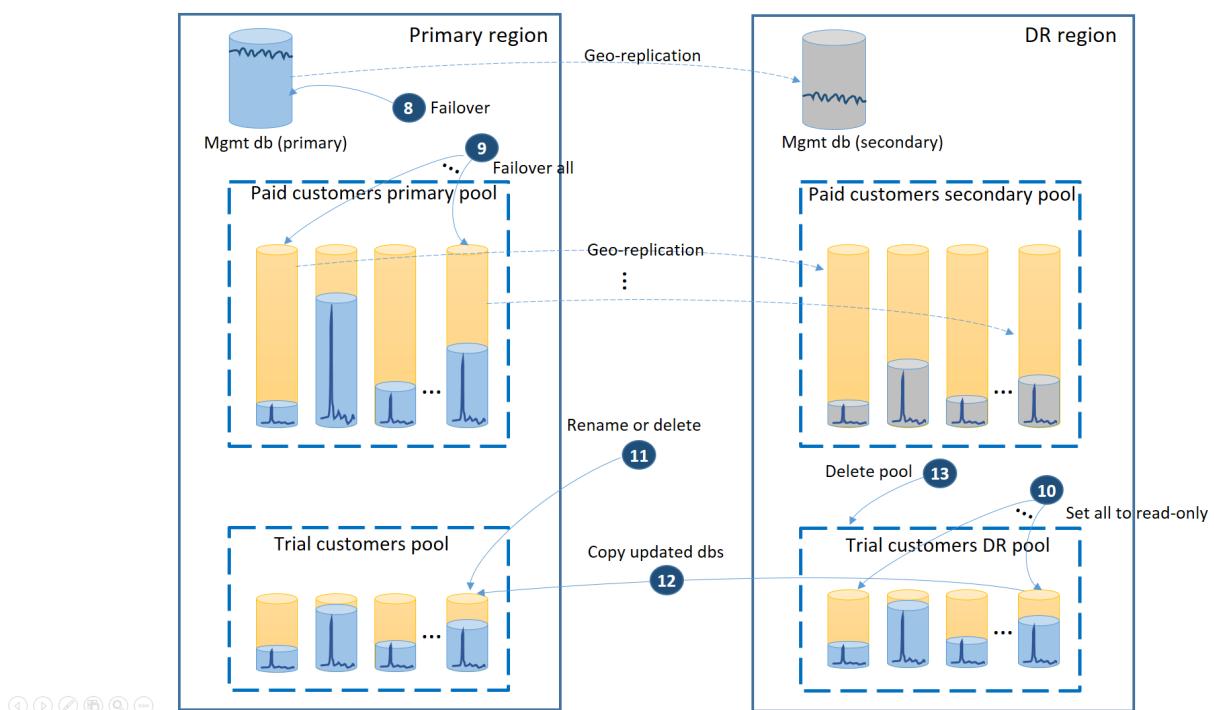
- Immediately fail over the management databases to the DR region (3).
- Change the application's connection string to point to the DR region. Now all new accounts and tenant

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.

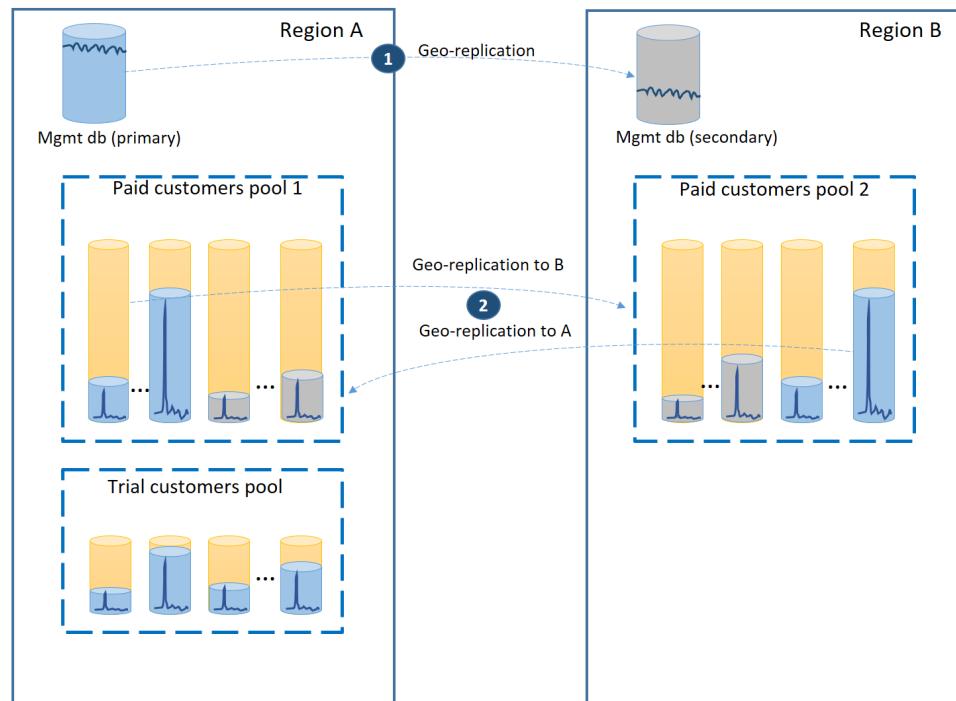
The key **benefit** of this strategy is that it provides the highest SLA for the paying customers. It also guarantees that the new trials are unblocked as soon as the trial DR pool is created. The **trade-off** is that this setup increases the total cost of the tenant databases by the cost of the secondary 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:

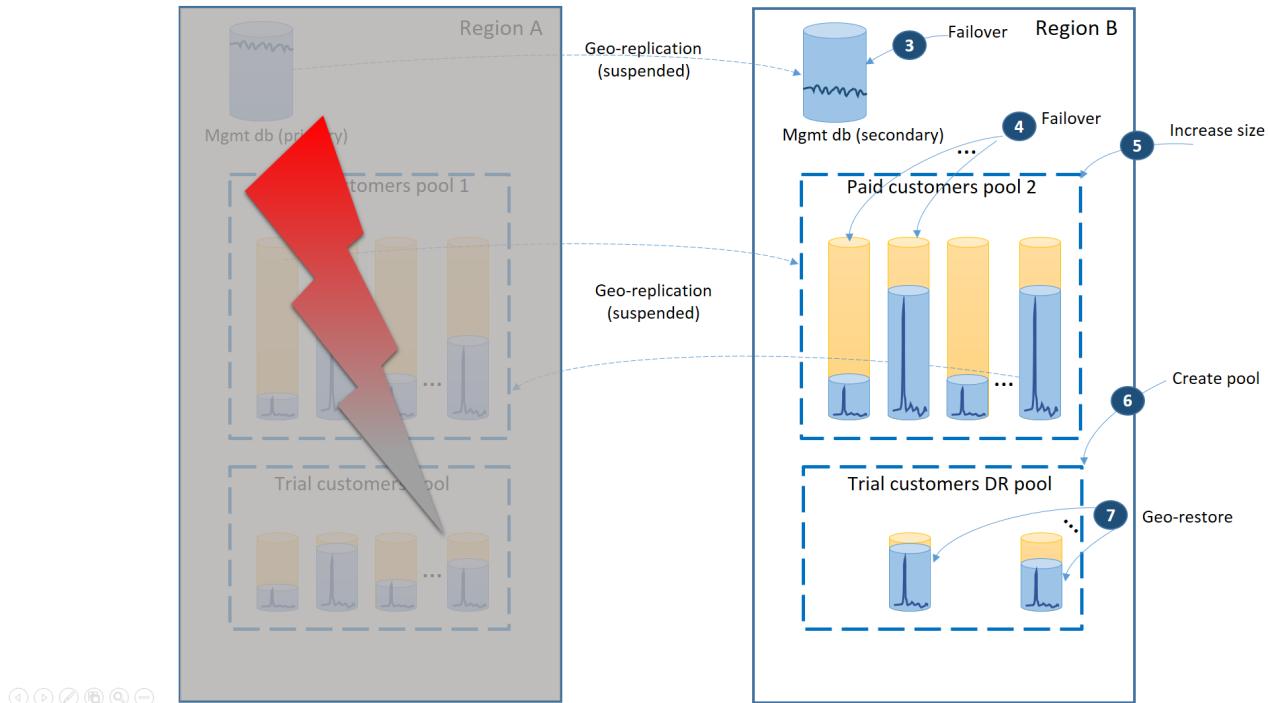


As in the previous scenarios, the management databases are quite active so configure them as single geo-replicated databases (1). This ensures the predictable performance of the new customer subscriptions, profile updates and other management operations. Region A is the primary region for the management 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.



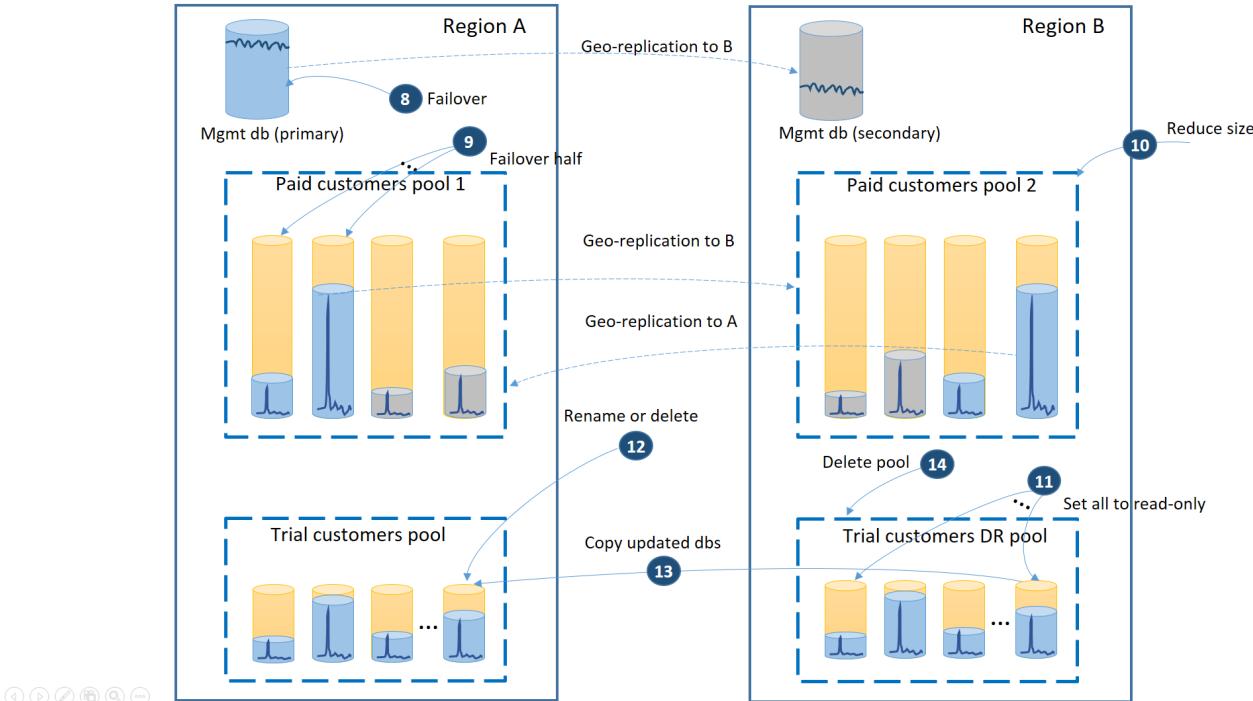
- Immediately fail over the management databases to region B (3).
- Change the application's connection string to point to the management databases in region B. Modify the management databases to make sure the new accounts and tenant databases are created in region B and the existing tenant databases 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](#).
- To learn about using automated backups for archiving, see [database copy](#).

# Managing rolling upgrades of cloud applications using SQL Database active geo-replication

9/25/2018 • 8 minutes to read • [Edit Online](#)

## NOTE

[Active geo-replication](#) is now available for all databases in all tiers.

Learn how to use [geo-replication](#) in SQL Database to enable rolling upgrades of your cloud application. Because upgrade is a disruptive operation, it 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 trade-offs of each option. For the purposes of this article we will use a simple application that consists of a web site connected to a single database as its data tier. Our goal is to upgrade version 1 of the application to version 2 without any significant impact on the end-user experience.

When evaluating the upgrade options you should consider the following factors:

- Impact on application availability during upgrades. How long the application function may be limited or degraded.
- Ability to roll back in case of an upgrade failure.
- Vulnerability of the application if an unrelated catastrophic failure occurs during the upgrade.
- Total dollar cost. This includes additional redundancy and incremental costs of the temporary components used by the upgrade process.

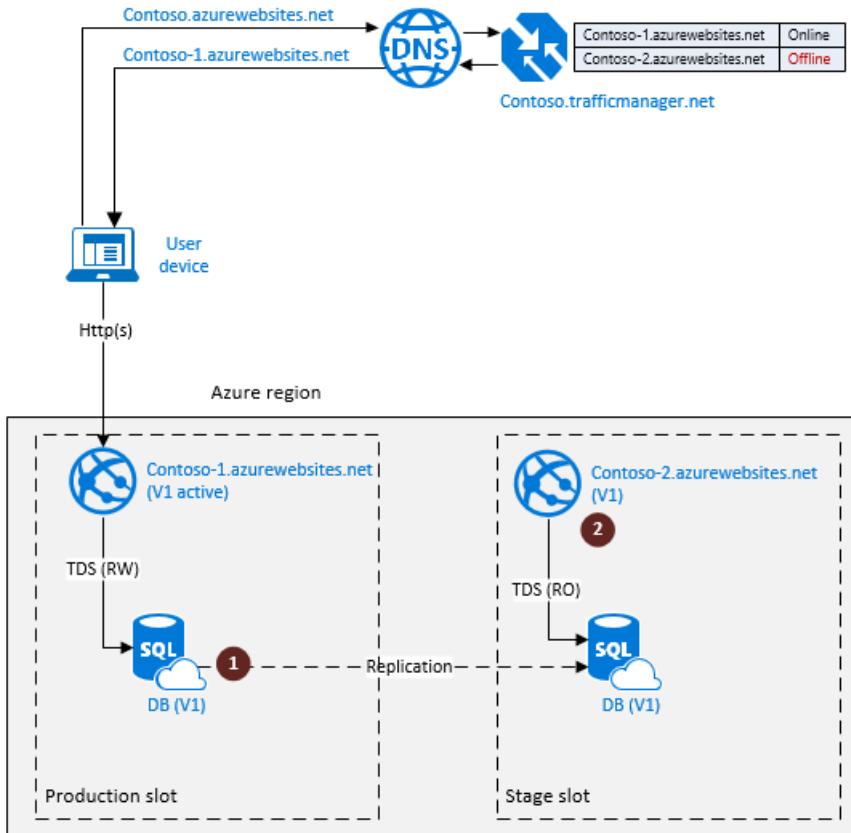
## Upgrading applications that rely on database backups for disaster recovery

If your application relies on automatic database backups and uses geo-restore for disaster recovery, it is usually deployed to a single Azure region. In this case the upgrade process involves creating a backup deployment of all application components involved in the upgrade. To minimize the end-user disruption you will leverage Azure Traffic Manager (ATM) with the failover profile. The following diagram illustrates the operational environment prior to the upgrade process. The endpoint *contoso-1.azurewebsites.net* represents a production slot of the application that needs to be upgraded. To enable the ability to roll back the upgrade, you need create a stage slot with a fully synchronized copy of the application. The following steps are required to prepare the application for the upgrade:

1. Create a stage slot for the upgrade. To do that create a secondary database (1) and deploy an identical web site in the same Azure region. Monitor the secondary to see if the seeding process is completed.
2. Create a failover profile in ATM with *contoso-1.azurewebsites.net* as online endpoint and *contoso-2.azurewebsites.net* as offline.

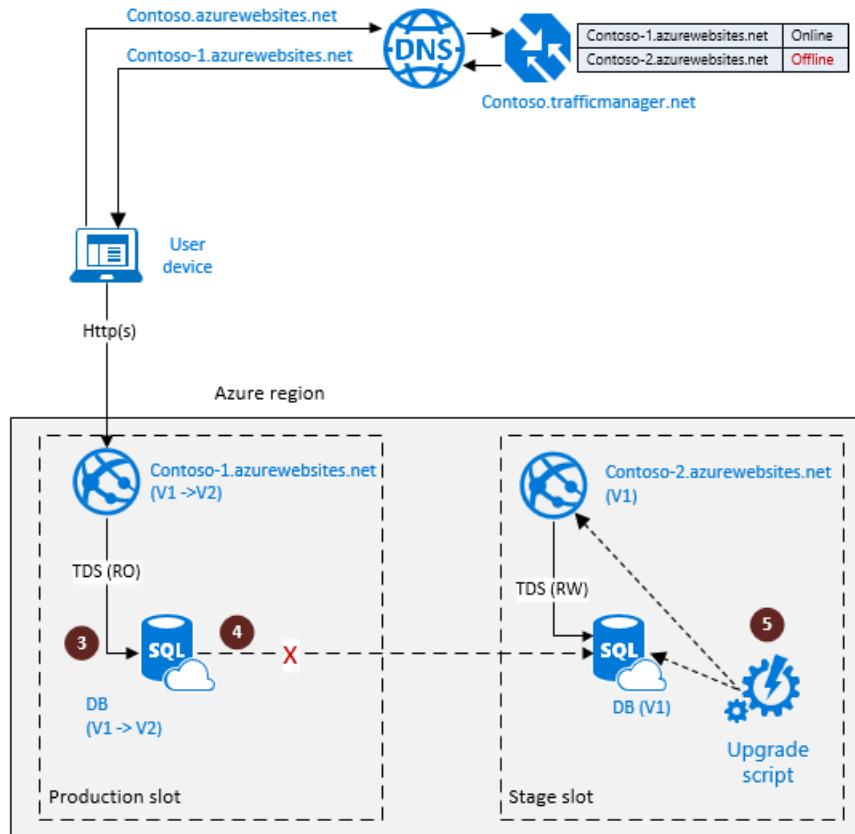
## NOTE

Note the preparation steps will not impact the application in the production slot and it can function in full access mode.



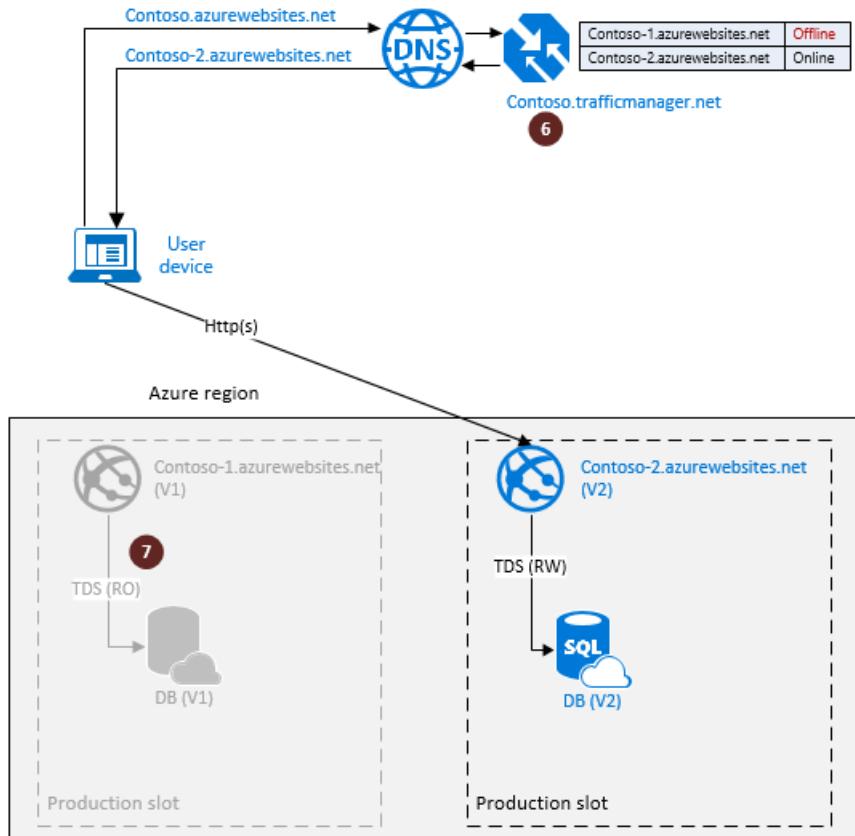
Once the preparation steps are completed the application is ready for the actual upgrade. The following diagram illustrates the steps involved in the upgrade process.

1. Set the primary database in the production slot to read-only mode (3). This will guarantee that the production instance of the application (V1) will remain read-only during the upgrade thus preventing the data divergence between the V1 and V2 database instances.
2. Disconnect the secondary database using the planned termination mode (4). It will create a fully synchronized independent copy of the primary database. This database will be upgraded.
3. Turn the primary database to read-write mode and run the upgrade script in the stage slot (5).



If the upgrade completed successfully you are now ready to switch the end users to the staged copy the application. It will now become the production slot of the application. This involves a few more steps as illustrated on the following diagram.

1. Switch the online endpoint in the ATM profile to `contoso-2.azurewebsites.net`, which points to the V2 version of the web site (6). It now becomes the production slot with the V2 application and the end-user traffic is directed to it.
2. If you no longer need the V1 application components so you can safely remove them (7).



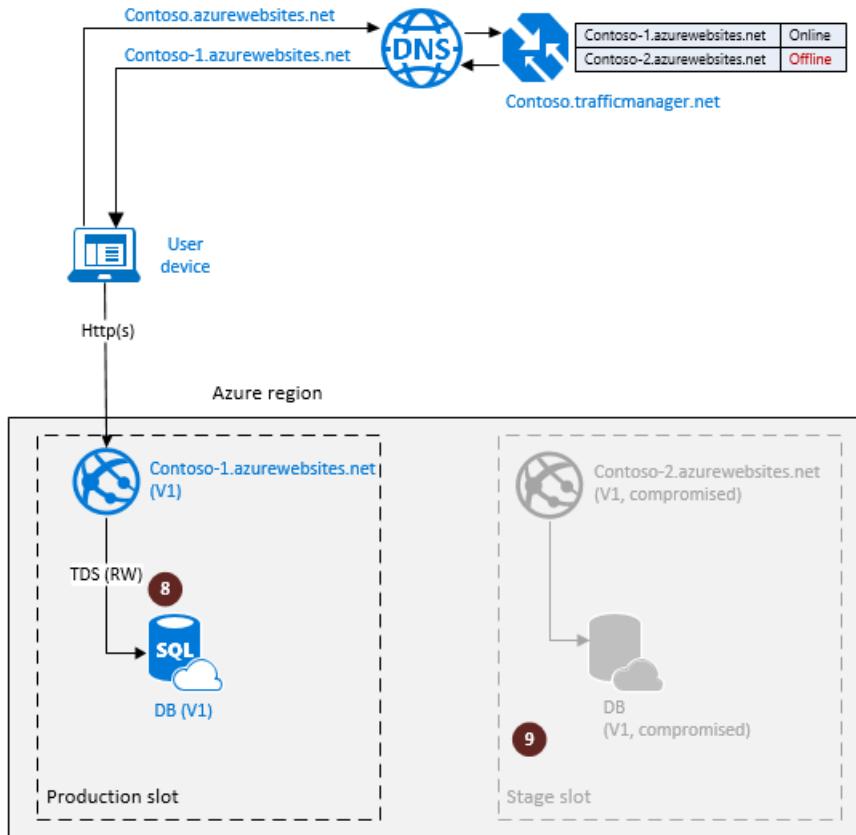
If the upgrade process is unsuccessful, for example due to an error in the upgrade script, the stage slot should be considered compromised. To roll back the application to the pre-upgrade state you simply revert the application in the production slot to full access. The steps involved are shown on the next diagram.

1. Set the database copy to read-write mode (8). This will restore the full V1 functionality in the production slot.
2. Perform the root cause analysis and remove the compromised components in the stage slot (9).

At this point the application is fully functional and the upgrade steps can be repeated.

#### NOTE

The rollback does not require changes in ATM profile as it already points to *contoso-1.azurewebsites.net* as the active endpoint.



The key **advantage** of this option is that you can upgrade an application in a single region using 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 will involve re-deployment of the application in a different region and restoring the database from backup using geo-restore. This process will result in significant downtime.

## Upgrading applications that rely on database geo-replication for disaster recovery

If your application leverages geo-replication for business continuity, it is deployed to at least two different regions with an active deployment in Primary region and a standby deployment in Backup region. In addition to the factors mentioned earlier, the upgrade process must 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 you will leverage Azure Traffic Manager (ATM) using the failover profile with one active and three backup endpoints. The following 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 slot of the application with full geographic redundancy. To enable the ability to roll back the upgrade, you need create a stage slot 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 stage slot needs to be geo-redundant as well. The following steps are required to prepare the application for the upgrade:

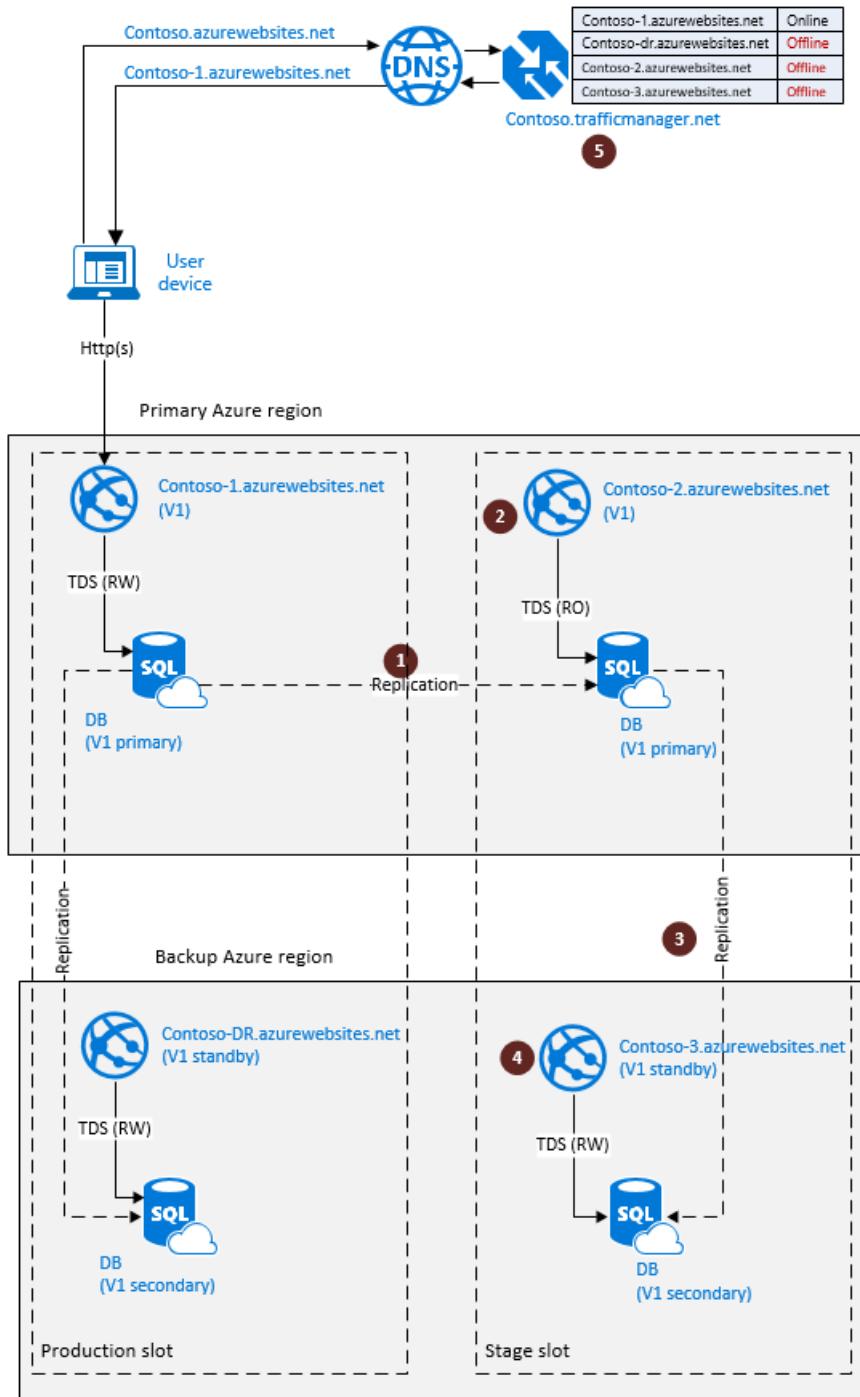
1. Create a stage slot for the upgrade. To do that create a secondary database (1) and deploy an identical copy of the web site in the same Azure region. Monitor the secondary to see if the seeding process is completed.
2. Create a geo-redundant secondary database in the stage slot by geo-replicating the secondary database to the backup region (this is called "chained geo-replication"). Monitor the backup secondary to see if the seeding

process is completed (3).

3. Create a standby copy of the web site in the backup region and link it to the geo-redundant secondary (4).
4. Add the additional endpoints `contoso-2.azurewebsites.net` and `contoso-3.azurewebsites.net` to the failover profile in ATM as offline endpoints (5).

**NOTE**

Note the preparation steps will not impact the application in the production slot and it can function in full access mode.

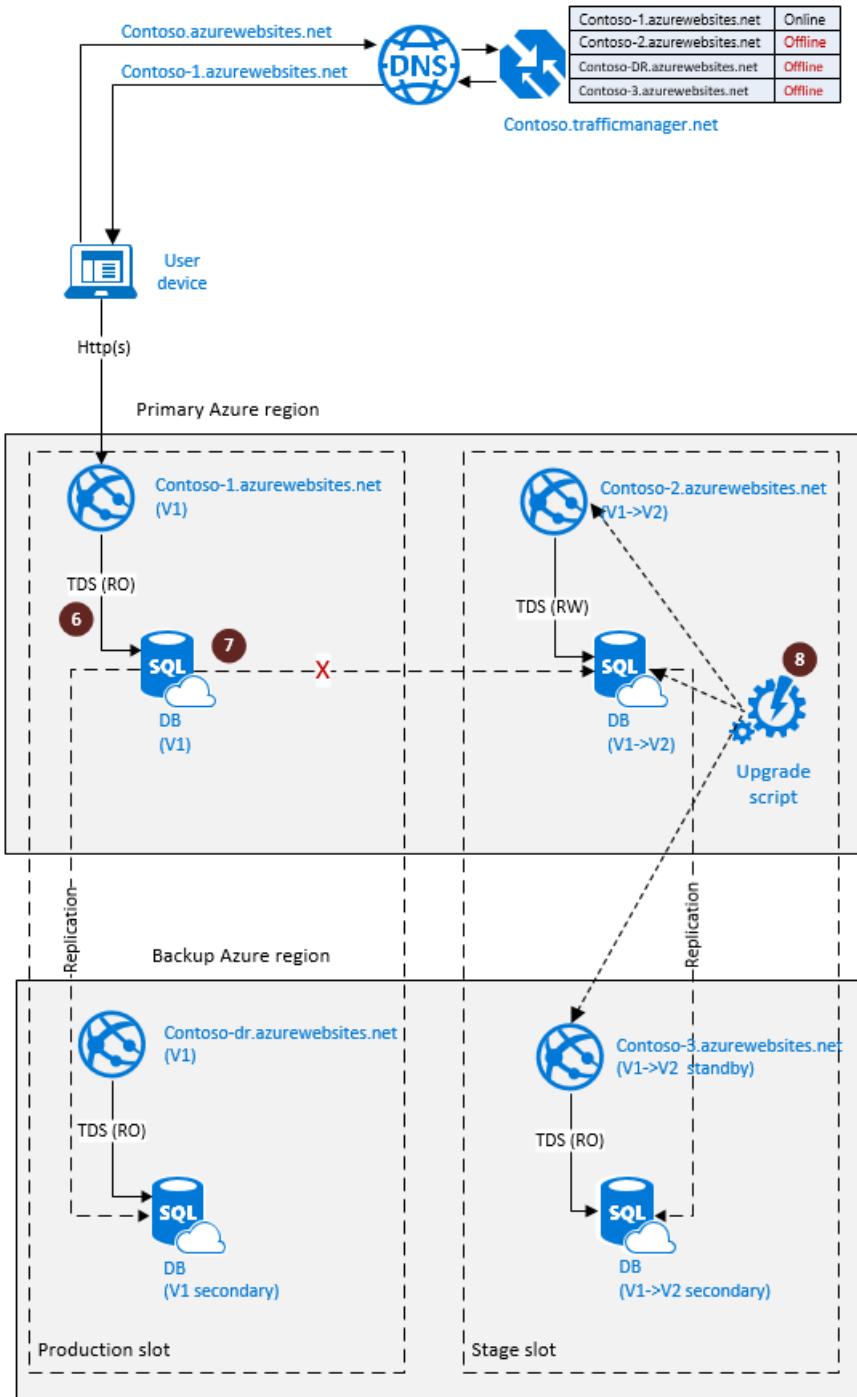


Once the preparation steps are completed, the stage slot is ready for the upgrade. The following diagram illustrates the upgrade steps.

1. Set the primary database in the production slot to read-only mode (6). This will guarantee that the production instance of the application (V1) will remain read-only during the upgrade thus preventing the data divergence

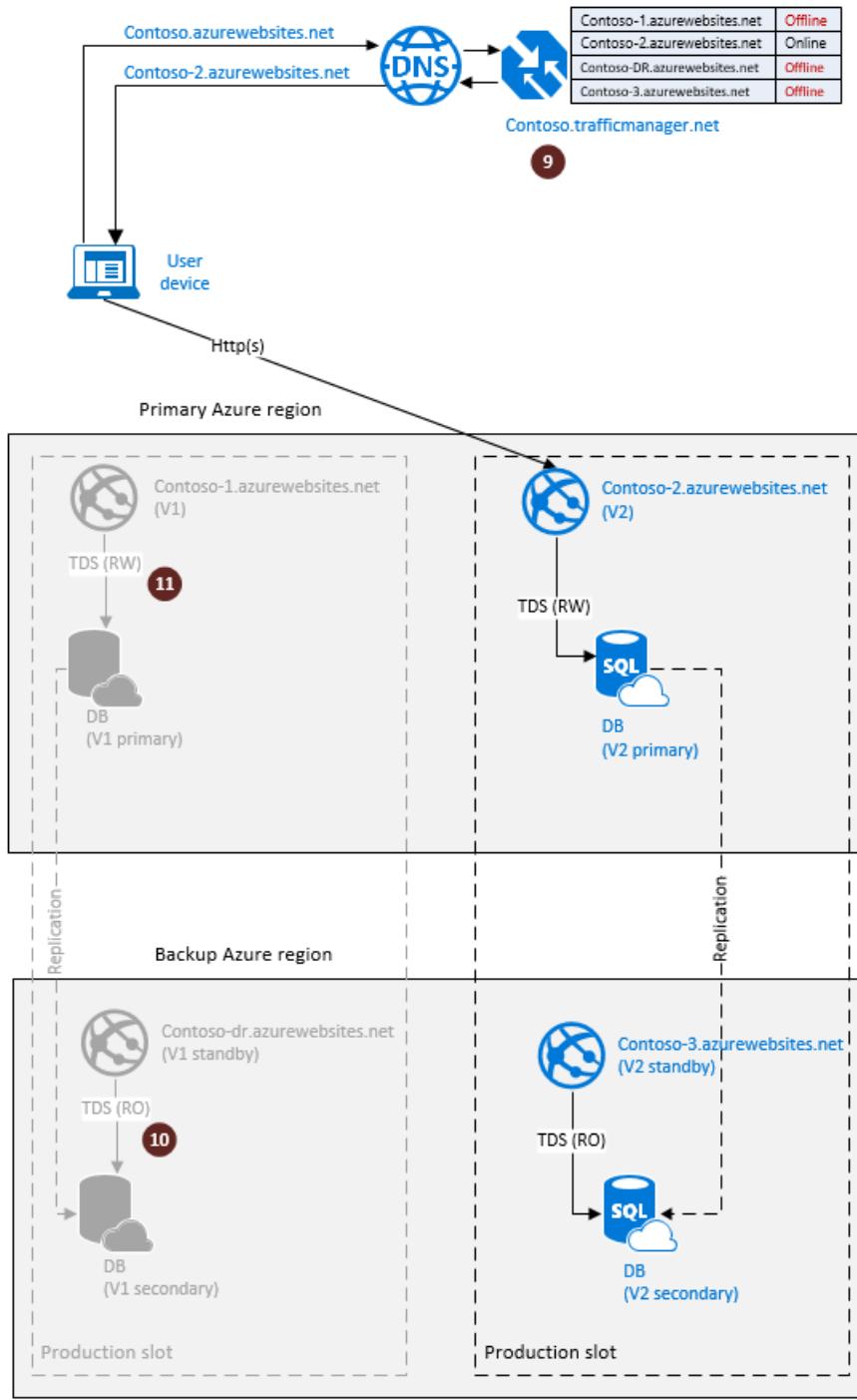
between the V1 and V2 database instances.

2. Disconnect the secondary database in the same region using the planned termination mode (7). It will create a fully synchronized independent copy of the primary database, which will automatically become a primary after the termination. This database will be upgraded.
3. Turn the primary database in the stage slot to read-write mode and run the upgrade script (8).



If the upgrade completed successfully you are now ready to switch the end users to the V2 version of the application. The following diagram illustrates the steps involved.

1. Switch the active endpoint in the ATM profile to `contoso-2.azurewebsites.net`, which now points to the V2 version of the web site (9). It now becomes a production slot with the V2 application and end-user traffic is directed to it.
2. If you no longer need the V1 application so you can safely remove it (10 and 11).



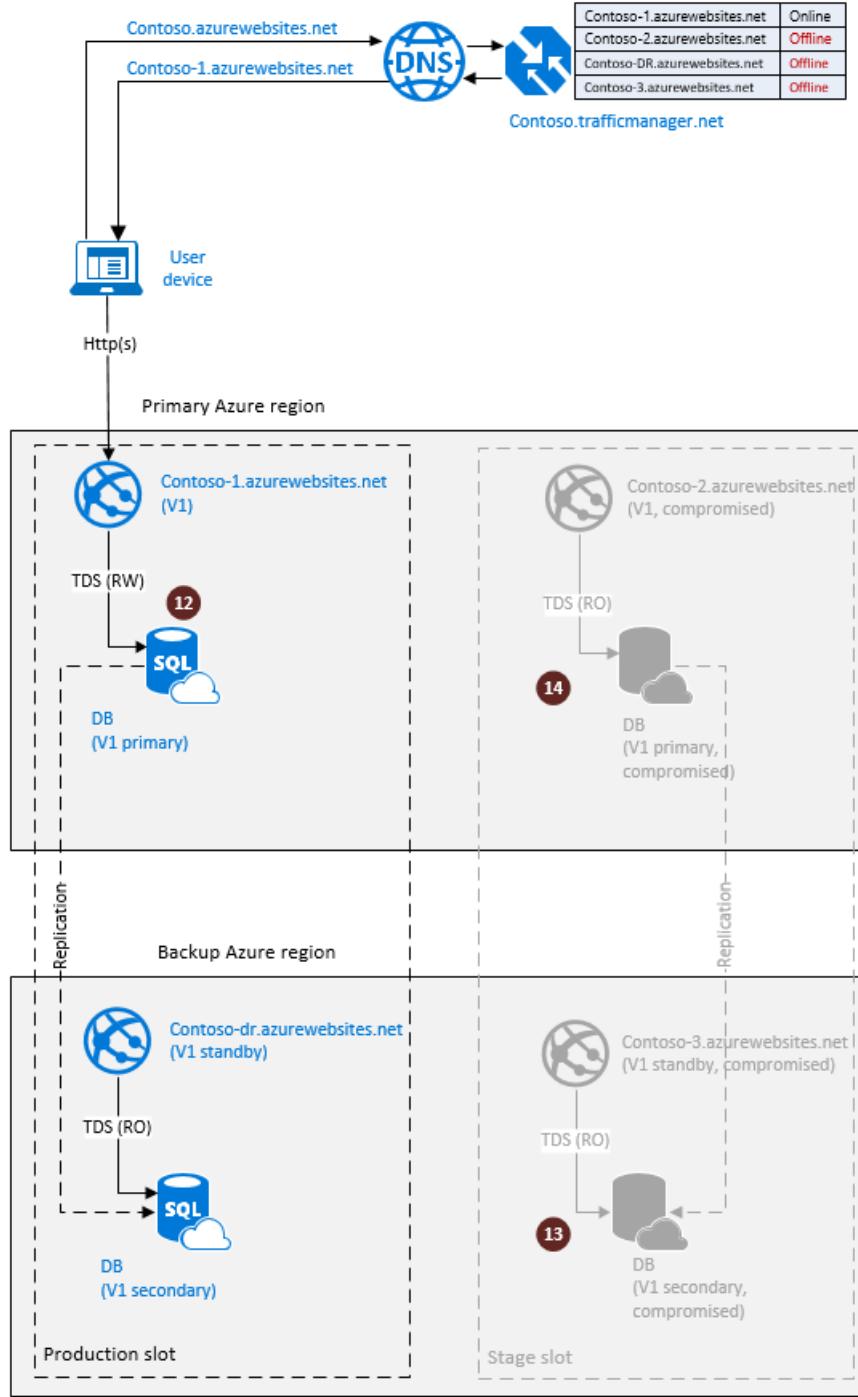
If the upgrade process is unsuccessful, for example due to an error in the upgrade script, the stage slot should be considered compromised. To roll back the application to the pre-upgrade state you simply revert to using the application in the production slot with full access. The steps involved are shown on the next diagram.

1. Set the primary database copy in the production slot to read-write mode (12). This will restore the full V1 functionality in the production slot.
2. Perform the root cause analysis and remove the compromised components in the stage slot (13 and 14).

At this point the application is fully functional and the upgrade steps can be repeated.

#### NOTE

The rollback does not require changes in ATM profile as it already points to *contoso-1.azurewebsites.net* as the active endpoint.



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 the dollar cost but they both focus on minimizing the time when the end user is limited to read-only operations. That time is directly defined by the duration of the upgrade script. It does not depend on the database size, the service tier you chose, the web site configuration and other factors that you cannot easily control. This is because all the preparation steps are decoupled from the upgrade steps and can be done without impacting the production application. The efficiency of the upgrade script is the key factor that determines the end-user experience during upgrades. So the best way you can improve it is by focusing 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 automated backups, see [SQL Database automated backups](#).
- To learn about using automated backups for recovery, see [restore a database from automated backups](#).
- To learn about faster recovery options, see [active geo-replication](#).

# An overview of Azure SQL Database security capabilities

10/30/2018 • 6 minutes to read • [Edit Online](#)

This article walks through the basics of securing the data tier of an application using Azure SQL Database. In particular, this article gets you started with resources for protecting data, controlling access, and proactive monitoring.

For a complete overview of security features available on all flavors of SQL, see the [Security Center for SQL Server Database Engine and Azure SQL Database](#). Additional information is also available in the [Security and Azure SQL Database technical white paper](#) (PDF).

## Protect data

### Encryption

SQL Database secures your data by providing encryption for data in motion with [Transport Layer Security](#), for data at rest with [Transparent Data Encryption](#), and for data in use with [Always Encrypted](#).

#### IMPORTANT

Azure SQL Database enforces encryption (SSL/TLS) at all times for all connections, which ensures all data is encrypted "in transit" between the database and the client. This will happen irrespective of the setting of **Encrypt** or **TrustServerCertificate** in the connection string.

In your application's connection string, ensure that you specify an encrypted connection and *not* to trust the server certificate (For the ADO.NET driver this is **Encrypt=True** and **TrustServerCertificate=False**). This helps to prevent your application from a man in the middle attack, by forcing the application to verify the server and enforcing encryption. If you obtain your connection string from the Azure portal, it will have the correct settings.

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

For other ways to encrypt your data, consider:

- [Cell-level encryption](#) to encrypt specific columns or even cells of data with different encryption keys.
- If you need a Hardware Security Module or Bring Your Own Key (BYOK) technology for Transparent Data Encryption, consider using [Azure SQL Transparent Data Encryption: Bring Your Own Key support](#).

### 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, PII, 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 SQL DB Data Discovery & Classification](#).

## Control access

SQL Database secures your data by limiting access to your database using firewall rules, authentication mechanisms requiring users to prove their identity, and authorization to data through role-based memberships and permissions, as well as through row-level security and dynamic data masking. For a discussion of the use of access control features in SQL Database, see [Control access](#).

#### **IMPORTANT**

Managing databases and logical 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](#).

### **Firewall and firewall rules**

To help protect your data, firewalls prevent all access to your database server until you specify which computers have permission using [firewall rules](#). The firewall grants access to databases based on the originating IP address of each request.

### **Authentication**

SQL database authentication refers to how you prove your identity when connecting to the database. SQL Database supports two types of authentication:

- **SQL Authentication**

This authentication method uses a username and password. When you created the logical 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. Use Active Directory authentication (integrated security) [whenever possible](#). 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.

### **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.

### **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). For more information, see [Row-Level security](#).

### **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](#).

# Proactive monitoring

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

## Auditing

SQL Database Auditing tracks database activities and helps you to maintain regulatory compliance, by recording database events to an audit log in your Azure Storage account. Auditing enables you to understand 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](#).

## Threat detection

Threat Detection complements auditing, by providing an additional layer of security intelligence built into the Azure SQL Database 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. Threat Detection 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. Threat Detection costs \$15/server/month. It is free for the first 60 days. For more information, see [Get started with SQL Database Threat Detection](#).

# 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](#).

# Security management

SQL Database helps you manage your data security by providing database scans and a centralized security dashboard using [SQL Vulnerability Assessment](#).

**SQL Vulnerability Assessment** is an easy to configure tool built into Azure SQL Database that can help you discover, track, and remediate potential database vulnerabilities. The assessment executes a vulnerability scan on your database, and generates a report that gives you visibility into your security state, including actionable steps to resolve security issues and enhance your database security. The assessment report can be customized for your environment, by setting an acceptable baseline for permission configurations, feature configurations, and database settings. This can help you to:

- Meet compliance requirements that require database scan reports.
- Meet data privacy standards.
- Monitor a dynamic database environment where changes are difficult to track.

For more information, see [SQL Vulnerability Assessment](#).

# 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](#).

# Advanced Threat Protection for Azure SQL Database

9/25/2018 • 3 minutes to read • [Edit Online](#)

SQL Advanced Threat Protection 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

SQL Advanced Threat Protection (ATP) provides a set of advanced SQL security capabilities, including Data Discovery & Classification, Vulnerability Assessment, and Threat Detection.

- [Data Discovery & Classification](#) (currently in preview) 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.
- [Threat Detection](#) 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. Threat Detection alerts provide details of the suspicious activity and recommend action on how to investigate and mitigate the threat.

Enable SQL ATP once to enable all of these included features. With one click, you can enable ATP on your entire database server, applying to all databases on the server.

ATP pricing aligns with Azure Security Center standard tier at \$15/node/month, where each protected SQL Database server is counted as one node. The first 60 days after enablement are considered a free trial period and are not charged. For more information, see the [Azure Security Center pricing page](#).

## Getting Started with ATP

The following steps get you started with ATP.

### 1. Enable ATP

Enable ATP by navigating to **Advanced Threat Protection** under the **Security** heading in your Azure SQL Database pane. To enable ATP for all databases on the server, click **Enable Advanced Threat Protection on the server**.

Home > ContosoCRMDB - Advanced Threat Protection

ContosoCRMDB - Advanced Threat Protection

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 Threat Protection

Auditing & Threat Detection

Search (Ctrl+ /)

Settings Feedback

Enable Advanced Threat Protection on the server

Data Discovery & Classification (preview)

Vulnerability Assessment

Threat Detection

#### NOTE

The cost of ATP is \$15/node/month, where a node is the entire SQL logical server. You are thus paying only once for protecting all databases on the server with ATP. The first 60 days are considered a free trial.

## 2. Configure Vulnerability Assessment

To start using Vulnerability Assessment, you need to configure a storage account where scan results are saved. To do so, click on the Vulnerability Assessment card.

Home > ContosoCRMDB - Advanced Threat Protection

ContosoCRMDB - Advanced Threat Protection

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 Threat Protection

Auditing & Threat Detection

Search (Ctrl+ /)

Settings Feedback

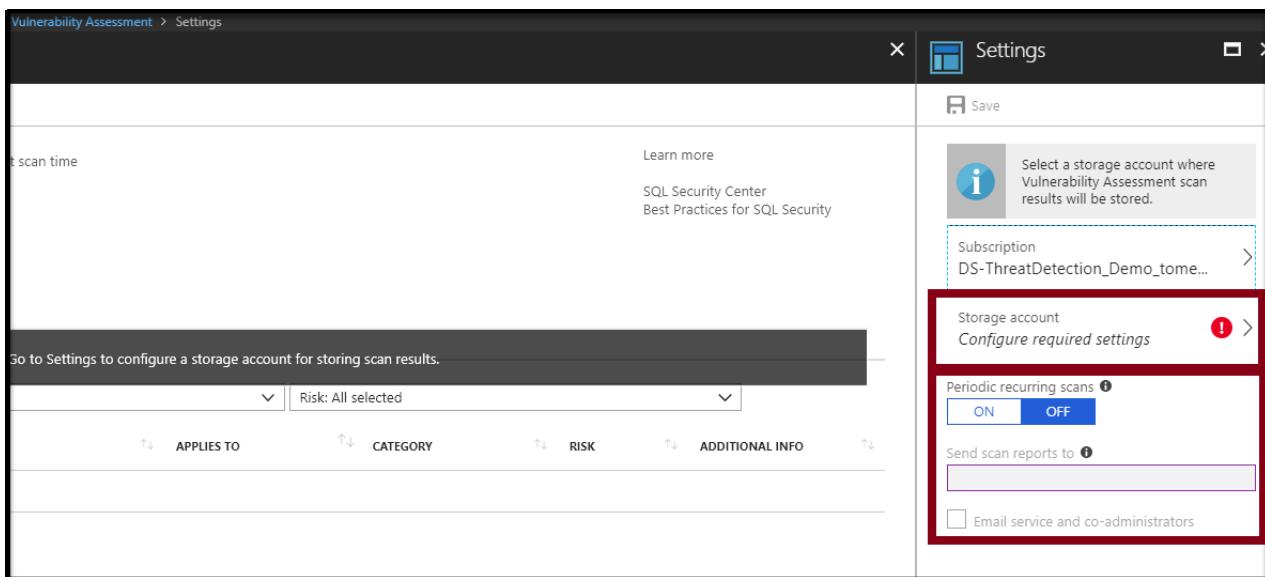
Successfully enabled Advanced Threat Protection on server

Data Discovery & Classification (preview)

Vulnerability Assessment

Threat Detection

Select or create a storage account for saving scan results. You can also turn on periodic recurring scans to configure Vulnerability Assessment to run automatic scans once per week. A scan result summary is sent to the email address(es) you provide.



### 3. Start classifying data, tracking vulnerabilities, and investigating threat alerts

Click the **Data Discovery and 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 **Threat Detection** 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.

### 4. Manage ATP settings on your SQL server

To view and manage Advanced Threat Protection settings, navigate to **Advanced Threat Protection** under the **Security** heading in your SQL server pane. On this page, you can enable or disable ATP, and modify Threat Detection settings for your entire SQL server.

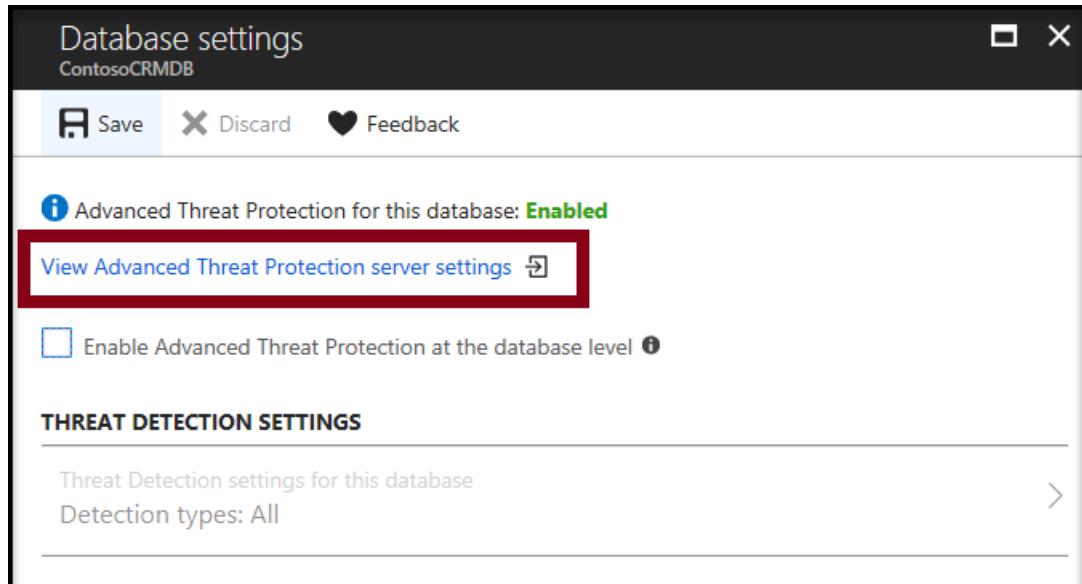
## 5. Manage ATP settings for a SQL database

To override ATP Threat Detection settings for a particular database, check the **Enable Advanced Threat Protection at the database level** checkbox. Use this option only if you have a particular requirement to receive separate threat detection alerts for the individual database, in place of or in addition to the alerts received for all databases on the server.

Once the checkbox is selected, click **Threat Detection settings for this database** and then configure the relevant settings for this database.

Advanced Threat Protection settings for your server can also be reached from the ATP database pane. Click

**Settings** in the main ATP pane, and then click **View Advanced Threat Protection server settings**.



## Next steps

- Learn more about [Data Discovery & Classification](#)
- Learn more about [Vulnerability Assessment](#)
- Learn more about [Threat Detection](#)
- Learn more about [Azure Security Center](#)

# Azure SQL Database Data Discovery and Classification

10/29/2018 • 5 minutes to read • [Edit Online](#)

Data Discovery & Classification (currently in preview) provides advanced capabilities built into Azure SQL Database for **discovering, classifying, labeling & protecting** 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 [SQL Advanced Threat Protection](#) (ATP) offering, which is a unified package for advanced SQL security capabilities. Data Discovery & Classification can be accessed and managed via the central SQL ATP portal.

## NOTE

This document relates to Azure SQL Database only. For SQL Server (on-prem), see [SQL Data Discovery and Classification](#).

## What is Data Discovery and 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 Threat Protection** under the Security heading in your Azure SQL Database pane. Click to enable Advanced Threat Protection, and then click on the **Data discovery & classification (preview)** card.

The screenshot shows the Azure portal interface for a SQL database named 'Clinic'. The 'Advanced Threat Protection' section is highlighted. The 'Data discovery & classification (preview)' card is open, displaying the following information:

- Data Discovery & Classification (preview)**: Shows 0 TOTAL classified columns.
- Vulnerability Assessment**: A donut chart indicates 5 TOTAL failures, with 0 HIGH RISK FAILURES, 0 MEDIUM RISK FAILURES, and 5 LOW RISK FAILURES.
- Failed Checks**: A table lists three failed security checks:

| SECURITY CHECK  | RISK   |
|---|--------|
| Server-level firewall rules should not grant excessi... | High   |
| Server-level firewall rules should be tracked and m...  | High   |
| Sensitive data columns should be classified             | Medium |
- Threat Detection**: Shows 0 TOTAL alerts, with 0 HIGH SEVERITY ALERTS and 0 MEDIUM SEVERITY ALERTS. A note states: "There are no alerts or recommendations to display."

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 (preview)

Overview

Classification

Classified columns: 15 / 109

Tables containing sensitive data: 5 / 12

Unique information types: 7

Label distribution:

- CONFIDENTIAL - GDPR
- HIGHLY CONFIDENTIAL
- CONFIDENTIAL
- GENERAL
- PUBLIC

Information type distribution:

- NAME
- BANKING
- CONTACT INFO
- CREDENTIALS
- FINANCIAL
- CREDIT CARD
- HEALTH

| Schema  | Table    | Column       | Information Type | Sensitivity Label   |
|---------|----------|--------------|------------------|---------------------|
| dbo     | ErrorLog | UserName     | Credentials      | Confidential        |
| SalesLT | Address  | AddressLine1 | Banking          | Highly confidential |
|         | Address  | AddressLine2 | Contact Info     | Confidential - GDPR |

4. To download a report in Excel format, click on the **Export** option in the top menu of the window.

Home > MayaDB - Data discovery & classification (preview)

MayaDB - Data discovery & classification (preview)

SQL database

Search (Ctrl+ /)

Export Feedback

Overview Classification

5. To begin classifying your data, click on the **Classification tab** at the top of the window.

Classification

6. 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 'Classification' tab in the 'Data discovery & classification (preview)' blade. A red box highlights the 'Accept selected recommendations' button. Below it, a table lists 15 columns with classification recommendations. The first column contains checkboxes, and the second column lists the schema (e.g., dbo, SalesLT). The table includes columns for SCHEMA, TABLE, COLUMN, INFORMATION TYPE, and SENSITIVITY LABEL.

|                                     | SCHEMA  | TABLE    | COLUMN       | INFORMATION TYPE | SENSITIVITY LABEL   |
|-------------------------------------|---------|----------|--------------|------------------|---------------------|
| <input type="checkbox"/>            | dbo     | ErrorLog | UserName     | Credentials      | Confidential        |
| <input checked="" type="checkbox"/> | SalesLT | Address  | AddressLine1 | Contact Info     | Confidential - GDPR |
| <input checked="" type="checkbox"/> | SalesLT | Address  | AddressLine2 | Contact Info     | Confidential - GDPR |
| <input checked="" type="checkbox"/> | SalesLT | Address  | City         | Contact Info     | Confidential - GDPR |
| <input checked="" type="checkbox"/> | SalesLT | Address  | PostalCode   | Contact Info     | Confidential - GDPR |
| <input checked="" type="checkbox"/> | SalesLT | Customer | FirstName    | Name             | Confidential - GDPR |

- To apply the selected recommendations, click on the blue **Accept selected recommendations** button.

A red box highlights the 'Accept selected recommendations' button in the 'Classification' tab. The button is located below a header that says '15 columns with classification recommendations'.

- 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 top menu bar of the 'Data discovery & classification (preview)' blade. A red box highlights the '+ Add classification' button. The menu bar also includes 'Save', 'Discard', and 'Feedback' buttons.

- 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.

8. 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.

## 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 |

## Automated/Programmatic classification

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](#)

You can also use REST APIs to programmatically manage classifications. The published REST APIs support the following operations:

- [Create Or Update](#) - Creates or updates the sensitivity label of a given column
- [Delete](#) - Deletes the sensitivity label of a given column
- [Get](#) - Gets the sensitivity label of a given column
- [List By Database](#) - Gets the sensitivity labels of a given database

## Next steps

- Learn more about [SQL Advanced Threat Protection](#).
- Consider configuring [Azure SQL Database Auditing](#) for monitoring and auditing access to your classified sensitive data.

# SQL Vulnerability Assessment service helps you identify database vulnerabilities

10/8/2018 • 5 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 [SQL Advanced Threat Protection](#) (ATP) offering, which is a unified package for advanced SQL security capabilities. Vulnerability Assessment can be accessed and managed via the central SQL ATP portal.

## 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 Threat Protection** under the Security heading in your Azure SQL Database pane. Click to enable Advanced Threat Protection, and then click on the **Vulnerability Assessment** card, which automatically opens the Vulnerability Assessment settings card.

Start by configuring a storage account where your scan results 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 Total passing checks Risk summary Last scan time Learn more

**0** **0** **High Risk** 0 **Medium Risk** 0 **Low Risk** 0

SQL Security Center Best Practices for SQL Security

**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 Total passing checks Risk summary Last scan time Learn more

**6** **42** **High Risk** 2 **Medium Risk** 3 **Low Risk** 1

Thu, 10 May 2018 19:41:29 UTC

SQL Security Center Best Practices for SQL Security

**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                   | <span style="color: red;">✖ FAIL</span>  |           |             |                |                     |
| 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 <a href="https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles">https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles</a> 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> <span style="float: right;"><span style="color: blue;">Run in Query Editor</span></span>  |           |             |                |                     |
| MICROSOFT RECOMMENDATION | Empty Set  |           |             |                |                     |
| RESULTS                  | IN BASELINE  | PRINCIPAL | ROLE        | PRINCIPAL TYPE | AUTHENTICATION TYPE |
|                          | ✗  | michael8  | db_ddladmin | SQL_USER       | NONE                |
|                          | ✗  | test1     | db_ddladmin | DATABASE_ROLE  | NONE                |

REMEDIATION

Remove members who should not have access to the database role

REMEDIATION 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                   | <span style="color: red;">X</span> 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 <a href="https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles">https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/database-level-roles</a> 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>   |           |             |                |
|                          | <a href="#">Run in Query Editor</a>  |           |             |                |
| MICROSOFT RECOMMENDATION | Empty Set  |           |             |                |
| RESULTS                  | IN BASELINE  | PRINCIPAL | ROLE        | PRINCIPAL TYPE |
|                          | <span style="color: red;">X</span>   | michael8  | db_ddladmin | SQL_USER       |
|                          | <span style="color: red;">X</span>   | test1     | db_ddladmin | DATABASE_ROLE  |

## 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 |             |   |          |   |                               |  |
| <span style="color: red;">3</span> <span style="color: red;">X</span> | <span style="color: green;">45</span> <span style="color: green;">✓</span> | <table border="1"> <tr> <td>High Risk</td> <td>0</td> </tr> <tr> <td>Medium Risk</td> <td>2</td> </tr> <tr> <td>Low Risk</td> <td>1</td> </tr> </table> | High Risk      | 0          | Medium Risk | 2 | Low Risk | 1 | Thu, 17 May 2018 09:50:22 UTC | <a href="#">SQL Security Center</a><br><a href="#">Best Practices for SQL Security</a> |
| High Risk   | 0  |   |                |            |             |   |          |   |                               |  |
| Medium Risk   | 2  |   |                |            |             |   |          |   |                               |  |
| Low Risk  | 1  |   |                |            |             |   |          |   |                               |  |

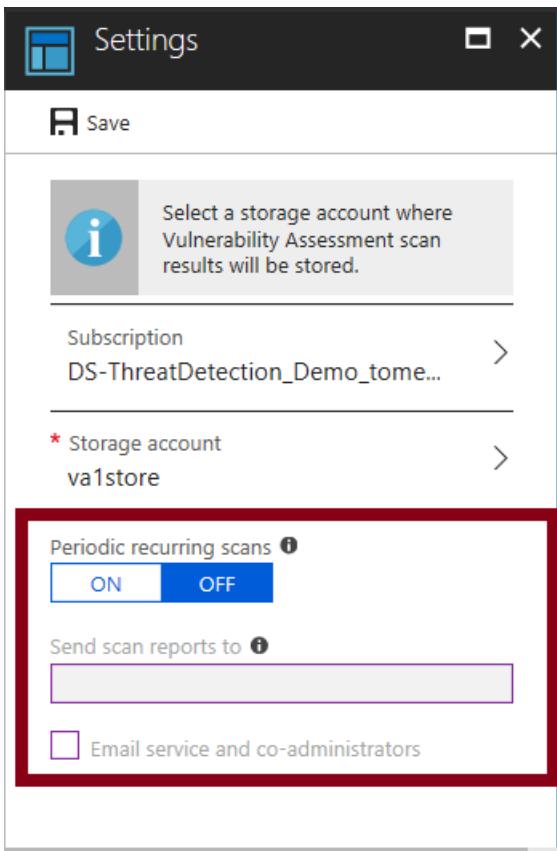
[Failed \(3\)](#) [Passed \(45\)](#)

| ID      | SECURITY CHECK   | APPLIES TO   | CATEGORY                 | STATUS  |
|---------|--|--------------|--------------------------|---|
| VA1281  | All memberships for user-defined roles should be intended                          | ContosoCRMDB | Auditing & Logging       | <span style="color: green;">✓ PASS (per custom baseline)</span> |
| VA2108  | Minimal set of principals should be members of fixed high impact database roles    | ContosoCRMDB | Authentication & Auth... | <span style="color: green;">✓ PASS (per custom baseline)</span> |
| VA20... | Server-level firewall rules should be tracked and maintained at a strict minimum   | master       | Surface area reduction   | <span style="color: green;">✓ PASS (per custom baseline)</span> |
| VA1020  | Server principal GUEST should not be a member of any role                          | ContosoCRMDB | Authentication & Auth... | <span style="color: green;">✓ PASS</span>                       |
| VA20... | Database-level firewall rules should not grant excessive access                    | ContosoCRMDB | Surface area reduction   | <span style="color: green;">✓ PASS</span>                       |
| VA20... | Database-level firewall rules should be tracked and maintained at a strict minimum | ContosoCRMDB | Surface area reduction   | <span style="color: green;">✓ PASS</span>                       |
| VA10... | Excessive permissions should not be granted to PUBLIC role on objects or columns   | ContosoCRMDB | Authentication & Auth... | <span style="color: green;">✓ PASS</span>                       |
| VA10... | Principal GUEST should not be granted permissions in the database                  | ContosoCRMDB | Authentication & Auth... | <span style="color: green;">✓ PASS</span>                       |

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

You can use Azure PowerShell cmdlets to programmatically manage your vulnerability assessments. The supported cmdlets are:

- [Update-AzureRmSqlDatabaseVulnerabilityAssessmentSettings](#)  
Updates the vulnerability assessment settings of a database
- [Get-AzureRmSqlDatabaseVulnerabilityAssessmentSettings](#)  
Returns the vulnerability assessment settings of a database
- [Clear-AzureRmSqlDatabaseVulnerabilityAssessmentSettings](#)  
Clears the vulnerability assessment settings of a database
- [Set-AzureRmSqlDatabaseVulnerabilityAssessmentRuleBaseline](#)  
Sets the vulnerability assessment rule baseline.
- [Get-AzureRmSqlDatabaseVulnerabilityAssessmentRuleBaseline](#)

Gets the vulnerability assessment rule baseline for a given rule.

- [Clear-AzureRmSqlDatabaseVulnerabilityAssessmentRuleBaseline](#)

Clears the vulnerability assessment rule baseline. First set the baseline before using this cmdlet to clear it.

- [Start-AzureRmSqlDatabaseVulnerabilityAssessmentScan](#)

Triggers the start of a vulnerability assessment scan

- [Get-AzureRmSqlDatabaseVulnerabilityAssessmentScanRecord](#)

Gets all vulnerability assessment scan record(s) associated with a given database.

- [Convert-AzureRmSqlDatabaseVulnerabilityAssessmentScan](#)

Converts vulnerability assessment scan results to an Excel file

For a script example, see [Azure SQL Vulnerability Assessment PowerShell support](#).

## Next steps

- Learn more about [SQL Advanced Threat Protection](#)
- Learn more about [Data Discovery & Classification](#)

# Azure SQL Database Threat Detection

10/25/2018 • 5 minutes to read • [Edit Online](#)

Azure SQL Threat Detection for [SQL Database](#) and [SQL Data Warehouse](#) detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases.

Threat Detection is part of the [SQL Advanced Threat Protection](#) (ATP) offering, which is a unified package for advanced SQL security capabilities. Threat Detection can be accessed and managed via the central SQL ATP 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 Threat Detection?

SQL Threat Detection 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. SQL Threat Detection integrates alerts with [Azure Security Center](#), which includes details of suspicious activity and recommend action on how to investigate and mitigate the threat. SQL Threat Detection 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.

## Set up threat detection for your database 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 Threat Protection**.
3. On the **Advanced Threat Protection** configuration page:
  - Enable Advanced Threat Protection on the server.
  - In **Threat Detection 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. The left sidebar lists various management options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Failover groups, Long-term backup retention, Active Directory admin, SQL databases, SQL elastic pools, Deleted databases, Import/Export history, Properties, Locks, and Automation script. Under SECURITY, the 'Advanced Threat Protection' option is highlighted with a red box. The main content area displays information about Advanced Threat Protection costs (15 USD/server/month) and its features (Threat Detection, Data Discovery & Classification, Vulnerability Assessment). It includes a trial period invitation and a toggle switch set to 'ON'. Below this is the 'THREAT DETECTION SETTINGS' section, which includes a 'Send alerts to' field containing 'samplecrmwedemo' with a checked checkbox for 'Email service and co-administrators'. There are also sections for 'Storage details' (samplecrmwedemo) and 'Threat Detection types' (All).

## Set up threat detection using PowerShell

For a script example, see [Configure auditing and threat detection using PowerShell](#).

## 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 | <a href="#">View the vulnerable SQL statement</a>  |
| Remediation steps   | <a href="#">Read more about SQL Injection threat and how to fix the vulnerable application code.</a>   |

- 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.

| DESCRIPTION  | COUNT | DETECTED BY | ENVIRONMENT | DATE     | STATE  | SEVERITY |
|--|-------|-------------|-------------|----------|--------|----------|
| Potential SQL Injection                              | 3     | Microsoft   | Azure       | 13/05/18 | Active | High     |
| Potential SQL Brute Force attempt                    | 1     | Microsoft   | Azure       | 09/04/18 | Active | High     |
| Attempted logon by a potentially harmful application | 1     | Microsoft   | Azure       | 09/04/18 | Active | High     |
| A possible vulnerability to SQL Injection            | 1     | Microsoft   | Azure       | 09/04/18 | Active | Medium   |
| Logon from an unusual location                       | 1     | Microsoft   | Azure       | 09/04/18 | Active | Medium   |
| Logon by an unfamiliar principal                     | 1     | Microsoft   | Azure       | 09/04/18 | Active | Medium   |

- 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             | <span style="color: red;">!</span> 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 | <a href="#">View the vulnerable SQL statement</a>  |
| REMEDIATION STEPS   | <a href="#">Read more about SQL Injection threat and how to fix the vulnerable application code.</a> |

## Explore threat detection alerts for your database in the Azure portal

SQL Database Threat Detection integrates its alerts with [Azure Security Center](#). A live SQL threat detection tiles within the database and SQL ATP blades in the Azure portal tracks the status of active threats.

Click **Threat detection alert** to launch the Azure Security Center alerts page and get an overview of active SQL threats detected on the database or data warehouse.

## Azure SQL Database Threat Detection alerts

Threat Detection 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 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.

## Next steps

- Learn more about [SQL Advanced Threat Protection](#).
- Learn more about [Azure SQL Database Auditing](#)
- Learn more about [Azure Security Center](#)
- For more information on pricing, see the [SQL Database Pricing page](#)

# Azure SQL Database Managed Instance Threat Detection

9/25/2018 • 4 minutes to read • [Edit Online](#)

SQL Threat Detection detects anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases in an Azure SQL Database Managed Instance.

## Overview

Threat Detection detects anomalous database activities indicating potential security threats to Managed Instance. Threat Detection is now in preview for Managed Instance.

Threat Detection 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 database activities. Threat Detection makes it simple to address potential threats to the Managed Instance without the need to be a security expert or manage advanced security monitoring systems. For a full investigation experience, it is recommended to enable Azure Managed Instance Auditing, which writes database events to an audit log in your Azure storage account.

SQL Threat Detection integrates alerts with [Azure Security Center](#), and, each protected Managed Instance is billed at the same price as Azure Security Center Standard tier, at \$15/node/month, where each protected Managed Instance is counted as one node.

## Set up Threat Detection for your Managed Instance 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 **Threat Detection**.
3. In the Threat Detection configuration page
  - Turn **ON** Threat detection.
  - 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.
4. Click **Save** to save the new or updated threat detection policy.

The screenshot shows the Azure portal interface for managing a SQL managed instance named 'td-demo-gp8c'. The left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, SETTINGS (Quick start, Connection strings, Active Directory admin), Threat Detection (which is selected and highlighted with a red box), and Pricing tier. The main content area displays Threat Detection settings. It shows a message about costs and a toggle switch set to 'ON'. Below this are sections for Storage details (with value 'haimbstoragewcus') and Threat Detection types (set to 'All'). A 'Send alerts to' section is shown, with a checkbox for 'Email service and co-administrators' which has a green checkmark and is also highlighted with a red box.

## Explore anomalous Managed Instance activities upon detection of a suspicious event

1. You receive an email notification upon detection of anomalous database activities.

The email provides information about the suspicious security event including the nature of the anomalous activities, database name, server name, and the event time. In addition, it provides information on possible causes and recommended actions to investigate and mitigate the potential threat to the Managed Instance.



2. 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 SQL threats detected on the Managed Instance's

database.

The screenshot shows the 'Security alerts' section of the Azure portal. At the top, there are filter and security center links. Below that, a summary bar indicates 3 alerts, all of which are marked as 'HIGH SEVERITY'. A timeline shows the alerts occurred between Monday and Thursday. The main table lists the following details for each alert:

| DESCRIPTION             | COUNT | DETECTED BY | ENVIRONMEN... | DATE     | STATE  | SEVERITY |
|-------------------------|-------|-------------|---------------|----------|--------|----------|
| Potential SQL Injection | 2     | Microsoft   | Azure         | 02/27/18 | Active | High     |
| Potential SQL Injection | 1     | Microsoft   | Azure         | 02/26/18 | Active | High     |
| Potential SQL Injection | 1     | Microsoft   | Azure         | 02/25/18 | Active | High     |

3. 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 common Web application security issues on the Internet. SQL injection 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

database2

[Learn more](#)

### General information

|                      |   |
|----------------------|---|
| DESCRIPTION          | Potential SQL Injection was detected on your database database2 on server   |
| DETECTION TIME       | Tuesday, February 27, 2018, 1:15:13 PM  |
| SEVERITY             | <span style="color: red;">!</span> High   |
| STATE                | Active  |
| ATTACKED RESOURCE    | database2   |
| SUBSCRIPTION         |   |
| DETECTED BY          |  Microsoft   |
| ACTION TAKEN         | Detected  |
| ENVIRONMENT          |  Azure   |
| RESOURCE TYPE        |  SQL Database  |
| DATABASE             | database2   |
| SERVER               |   |
| PRINCIPAL NAME       | cloudSA   |
| APPLICATION          | .Net SqlClient Data Provider  |
| IP ADDRESS           | 10.0.1.5  |
| VULNERABLE STATEMENT | <pre>select * from sys.databases where database_id like '' or 1 = 1 --' and family = 'second sql test in prod - test in prod with UI policy with database2'</pre> |

## Managed Instance Threat Detection alerts

Threat Detection for Managed Instance 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 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. It means that 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 a Managed Instance, where someone has logged on to the Managed Instance from an unusual geographical location. In some cases, the alert detects a legitimate action (a new application or developer's maintenance operation). In other cases, the alert detects a malicious action (former employee, external attacker, and so on).
- **Access from unusual Azure data center:** This alert is triggered when there is a change in the access pattern to Managed Instance, where someone has logged on to the Managed Instance from an Azure Data Center that was not seen accessing this Managed Instance during the recent period. In some cases, the alert detects a legitimate action (your new application in Azure, Power BI, Azure SQL Query Editor, and so on). 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 Managed Instance server, where someone has logged on to the Managed Instance using an unusual principal (SQL user). In some cases, the alert detects a legitimate action (new application developer's maintenance operation). 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 a brute force attack.

## Next steps

- Learn about Managed Instance, see [What is a Managed Instance](#)
- Learn more about [Managed Instance Auditing](#)
- Learn more about [Azure Security Center](#)

# Azure SQL Database and SQL Data Warehouse firewall rules

10/23/2018 • 11 minutes to read • [Edit Online](#)

Microsoft Azure [SQL Database](#) and [SQL Data Warehouse](#) provide 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.

## 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**. Please see the following article on [connecting to a Managed Instance](#) for more information about the networking configuration needed.

## Virtual network rules as alternatives to IP rules

In addition to IP rules, the firewall also manages *virtual network rules*. Virtual network rules are based on Virtual Network service endpoints. Virtual network rules might be preferable to IP rules in some cases. To learn more, see [Virtual Network service endpoints and rules for Azure SQL Database](#).

## Overview

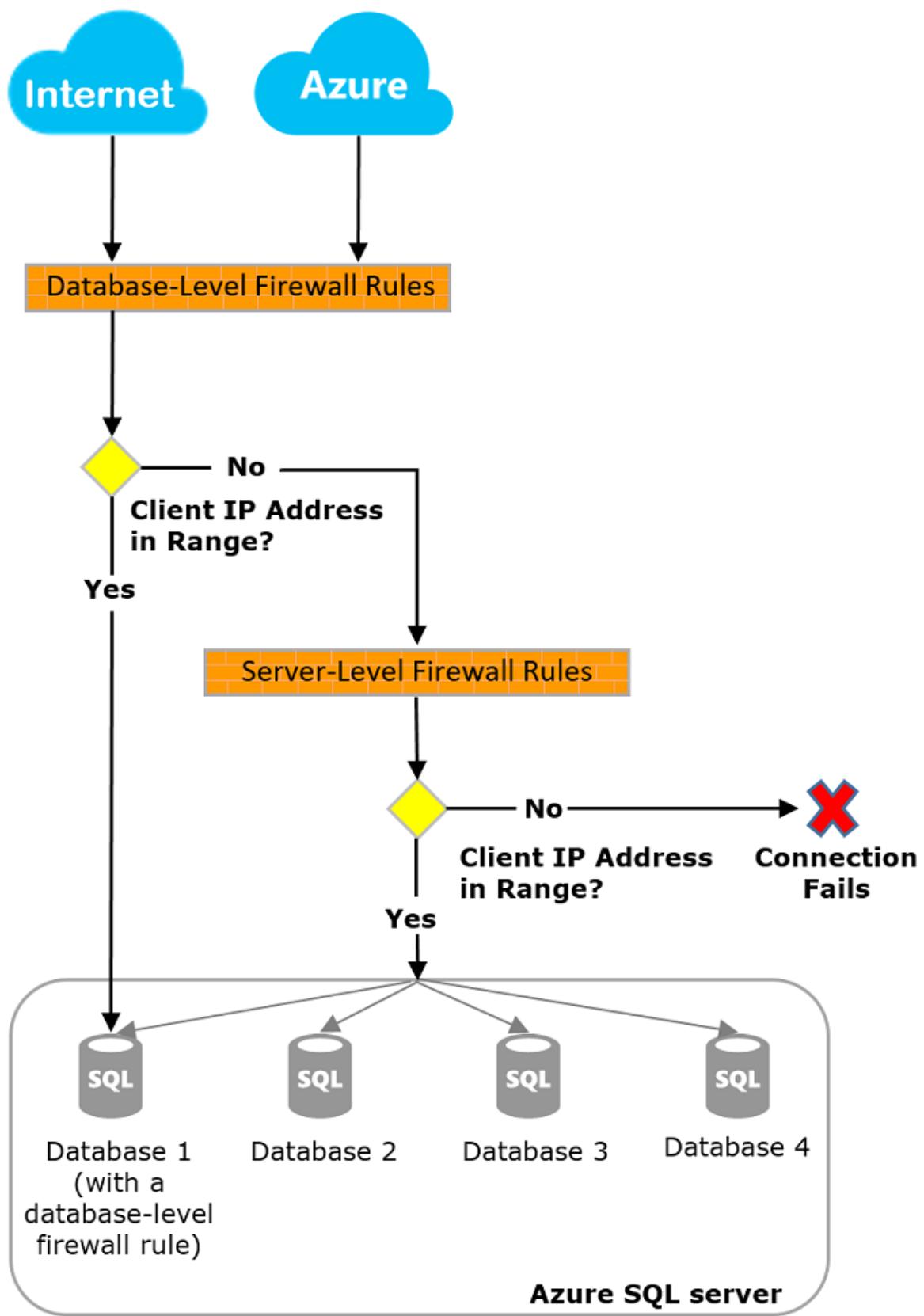
Initially, all Transact-SQL access to your Azure SQL server is blocked by the firewall. To begin using your Azure SQL server, you must specify one or more server-level firewall rules that enable access to your Azure SQL server. Use the firewall rules to specify which IP address ranges from the Internet are allowed, and whether Azure applications can attempt to connect to your Azure SQL server.

To selectively grant access to just one of the databases in your Azure SQL server, you must create a database-level rule for the required database. Specify an IP address range for the database firewall rule that is beyond the IP address range specified in the server-level firewall rule, and ensure that the IP address of the client falls in the range specified in the database-level rule.

## IMPORTANT

SQL Data Warehouse only supports server-level firewall rules and does not support database-level firewall rules.

Connection attempts from the Internet and Azure must first pass through the firewall before they can reach your Azure SQL server or SQL Database, as shown in the following diagram:



- **Server-level firewall rules:**

These rules enable clients to access your entire Azure SQL server, that is, all the databases within the same logical server. These rules are stored in the **master** database. Server-level firewall rules can be configured by using the portal or by using Transact-SQL statements. To create server-level firewall rules using the Azure portal or PowerShell, you must be the subscription owner or a subscription contributor. To create a server-level firewall rule using Transact-SQL, you must connect to the SQL Database instance as the server-level principal login or the Azure Active Directory administrator (which means that a server-level firewall rule must first be created by a user with Azure-level permissions).

- **Database-level firewall rules:**

These rules enable clients to access certain (secure) databases within the same logical server. You can create these rules for each database (including the **master** database) and they are stored in the individual databases. Database-level firewall rules for master and user databases can only be created and managed by using Transact-SQL statements and only after you have configured the first server-level firewall. If you specify an IP address range in the database-level firewall rule that is outside the range specified in the server-level 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 firewall rules for a database. For more information on configuring database-level firewall rules, see the example later in this article and see [sp\\_set\\_database\\_firewall\\_rule \(Azure SQL Database\)](#).

### Recommendation

Microsoft recommends using database-level firewall rules whenever possible to enhance security and to make your database more portable. Use server-level firewall rules for administrators and when you have many databases that have the same access requirements and you don't want to spend time configuring each database individually.

#### IMPORTANT

Windows Azure SQL Database supports a maximum of 128 firewall rules.

#### NOTE

For information about portable databases in the context of business continuity, see [Authentication requirements for disaster recovery](#).

### Connecting from the Internet

When a computer attempts to connect to your database server from the Internet, the firewall first checks the originating IP address of the request against the database-level firewall rules, for the database that the connection is requesting:

- If the IP address of the request is within one of the ranges specified in the database-level firewall rules, the connection is granted to the SQL Database that contains the rule.
- If the IP address of the request is not within one of the ranges specified in the database-level firewall rule, the server-level firewall rules are checked. If the IP address of the request is within one of the ranges specified in the server-level firewall rules, the connection is granted. Server-level firewall rules apply to all SQL databases on the Azure SQL server.
- If the IP address of the request is not within the ranges specified in any of the database-level or server-level firewall rules, the connection request fails.

#### NOTE

To access Azure SQL Database from your local computer, ensure the firewall on your network and local computer allows outgoing communication on TCP port 1433.

### Connecting from Azure

To allow applications from Azure to connect to your Azure SQL server, Azure connections must be enabled. When an application from Azure attempts to connect to your database server, the firewall verifies that Azure connections are allowed. A firewall setting with starting and ending address equal to 0.0.0.0 indicates these connections are allowed. If the connection attempt is not allowed, the request does not reach the Azure SQL Database server.

**IMPORTANT**

This option configures the firewall to allow all connections from Azure including connections from the subscriptions of other customers. When selecting this option, make sure your login and user permissions limit access to only authorized users.

## Creating and managing firewall rules

The first server-level firewall setting can be created using the [Azure portal](#) or programmatically using [Azure PowerShell](#), [Azure CLI](#), or the [REST API](#). Subsequent server-level firewall rules can be created and managed using these methods, and through Transact-SQL.

**IMPORTANT**

Database-level firewall rules can only be created and managed using Transact-SQL.

To improve performance, server-level 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.

## Manage firewall rules using the Azure portal

To set a server-level firewall rule in the Azure portal, you can either go to the Overview page for your Azure SQL database or the Overview page for your Azure Database logical server.

**TIP**

For a tutorial, see [Create a DB using the Azure portal](#).

### From database overview page

1. To set a server-level firewall rule from the database overview page, click **Set server firewall** on the toolbar as shown in the following image: The **Firewall settings** page for the SQL Database server opens.

- Click **Add client IP** on the toolbar to add the IP address of the computer you are currently using and then click **Save**. A server-level firewall rule is created for your current IP address.

### From server overview page

The overview page for your server opens, showing you the fully qualified server name (such as **mynewserver20170403.database.windows.net**) and provides options for further configuration.

- To set a server-level rule from server overview page, click **Firewall** in the left-hand menu under **Settings**:
- Click **Add client IP** on the toolbar to add the IP address of the computer you are currently using and then click **Save**. A server-level firewall rule is created for your current IP address.

## Manage firewall rules using Transact-SQL

| CATALOG VIEW OR STORED PROCEDURE     | LEVEL  | DESCRIPTION                                      |
|--------------------------------------|--------|--|
| <a href="#">sys.firewall_rules</a>   | Server | Displays the current server-level firewall rules |
| <a href="#">sp_set_firewall_rule</a> | Server | Creates or updates server-level firewall rules   |

| CATALOG VIEW OR STORED PROCEDURE                 | LEVEL     | DESCRIPTION  |
|--|-----------|--|
| <a href="#">sp_delete_firewall_rule</a>          | Server    | Removes server-level firewall rules                  |
| <a href="#">sys.database_firewall_rules</a>      | Database  | Displays the current database-level firewall rules   |
| <a href="#">sp_set_database_firewall_rule</a>    | Database  | Creates or updates the database-level firewall rules |
| <a href="#">sp_delete_database_firewall_rule</a> | Databases | Removes database-level firewall rules                |

The following examples review the existing rules, enable a range of IP addresses on the server Contoso, and deletes a firewall rule:

```
SELECT * FROM sys.firewall_rules ORDER BY name;
```

Next, add a 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 firewall rule, execute the `sp_delete_firewall_rule` stored procedure. The following example deletes the rule named `ContosoFirewallRule`:

```
EXECUTE sp_delete_firewall_rule @name = N'ContosoFirewallRule'
```

## Manage firewall rules using Azure PowerShell

| CMDLET  | LEVEL  | DESCRIPTION  |
|---|--------|--|
| <a href="#">Get-AzureRmSqlServerFirewallRule</a>    | Server | Returns the current server-level firewall rules                  |
| <a href="#">New-AzureRmSqlServerFirewallRule</a>    | Server | Creates a new server-level firewall rule                         |
| <a href="#">Set-AzureRmSqlServerFirewallRule</a>    | Server | Updates the properties of an existing server-level firewall rule |
| <a href="#">Remove-AzureRmSqlServerFirewallRule</a> | Server | Removes server-level firewall rules                              |

The following example sets a server-level firewall rule using PowerShell:

```
New-AzureRmSqlServerFirewallRule -ResourceGroupName "myResourceGroup" ` 
-ServerName $servername ` 
-FirewallRuleName "AllowSome" -StartIpAddress "0.0.0.0" -EndIpAddress "0.0.0.0"
```

**TIP**

For PowerShell examples in the context of a quick start, see [Create DB - PowerShell](#) and [Create a single database and configure a firewall rule using PowerShell](#)

## Manage firewall rules using Azure CLI

| CMDLET   | LEVEL  | DESCRIPTION                          |
|--|--------|--------------------------------------|
| <a href="#">az sql server firewall-rule create</a> | Server | Creates a server firewall rule       |
| <a href="#">az sql server firewall-rule list</a>   | Server | Lists the firewall rules on a server |
| <a href="#">az sql server firewall-rule show</a>   | Server | Shows the detail of a firewall rule  |
| <a href="#">az sql server firewall-rule update</a> | Server | Updates a firewall rule              |
| <a href="#">az sql server firewall-rule delete</a> | Server | Deletes a firewall rule              |

The following example sets a server-level firewall rule using the Azure CLI:

```
az sql server firewall-rule create --resource-group myResourceGroup --server $servername \
-n AllowYourIp --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0
```

**TIP**

For an Azure CLI example in the context of a quick start, see [Create DB - Azure CLI](#) and [Create a single database and configure a firewall rule using the Azure CLI](#)

## Manage firewall rules using REST API

| API  | LEVEL  | DESCRIPTION                                      |
|--|--------|--|
| <a href="#">List Firewall Rules</a>            | Server | Displays the current server-level firewall rules |
| <a href="#">Create or Update Firewall Rule</a> | Server | Creates or updates server-level firewall rules   |
| <a href="#">Delete Firewall Rule</a>           | Server | Removes server-level firewall rules              |
| <a href="#">Get Firewall Rules</a>             | Server | Gets server-level firewall rules                 |

## Server-level firewall rule versus a database-level firewall rule

Q. Should users of one database be fully isolated from another database? If yes, grant access using database-level firewall rules. This avoids using server-level firewall rules, which permit access through the firewall to all databases, reducing the depth of your defenses.

Q. Do users at the IP address's need access to all databases? Use server-level firewall rules to reduce the number of times you must configure firewall rules.

Q. Does the person or team configuring the firewall rules only have access through the Azure portal, PowerShell, or the REST API? You must use server-level firewall rules. Database-level firewall rules can only be configured using Transact-SQL.

Q. Is the person or team configuring the firewall rules prohibited from having high-level permission at the database level? Use server-level firewall rules. Configuring database-level firewall rules using Transact-SQL, requires at least `CONTROL DATABASE` permission at the database level.

Q. Is the person or team configuring or auditing the firewall rules, centrally managing firewall rules for many (perhaps 100s) of databases? This selection depends upon your needs and environment. Server-level firewall rules might be easier to configure, but scripting can configure rules at the database-level. And even if you use server-level firewall rules, you might need to audit the database-firewall rules, to see if users with `CONTROL` permission on the database have created database-level firewall rules.

Q. Can I use a mix of both server-level and database-level firewall rules? Yes. Some users, such as administrators might need server-level firewall rules. Other users, such as users of a database application, might need database-level firewall rules.

## Troubleshooting the database firewall

Consider the following points when access to the Microsoft Azure SQL Database service does not behave as you expect:

- **Local firewall configuration:**

Before your computer can access Azure SQL Database, you may need to create a firewall exception on your computer for TCP port 1433. If you are making 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 (NAT):**

Due to NAT, the IP address used by your computer to connect to Azure SQL Database may be different than the IP address shown in your computer IP configuration settings. To view the IP address your computer is using to connect to Azure, log in to the portal and navigate to the **Configure** tab on the server that hosts your database. Under the **Allowed IP Addresses** section, the **Current Client IP Address** is displayed. Click **Add** to the **Allowed IP Addresses** to allow this computer to access the server.

- **Changes to the allow list have not taken effect yet:**

There may be as much as a five-minute delay for changes to the Azure SQL Database firewall configuration to take effect.

- **The login is not authorized or an incorrect password was used:**

If a login does not have permissions on the Azure SQL Database server or the password used is incorrect, the connection to the Azure SQL Database server is denied. Creating a firewall setting only provides clients with an opportunity to attempt connecting to your server; each client must provide the necessary security credentials. For more information about preparing logins, see [Managing Databases, Logins, and Users in Azure SQL Database](#).

- **Dynamic IP address:**

If you have an Internet connection with dynamic IP addressing and you are having trouble getting through the firewall, you could try one of the following solutions:

- Ask your Internet Service Provider (ISP) for the IP address range assigned to your client computers

that access the Azure SQL Database server, and then add the IP address range as a firewall rule.

- Get static IP addressing instead for your client computers, and then add the IP addresses as firewall rules.

## Next steps

- For a quick start on creating a database and a server-level firewall rule, see [Create an Azure SQL database](#).
- For help in connecting to an Azure SQL database from open source or third-party applications, see [Client quick-start 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 security, see [Securing your database](#)

# Use Virtual Network service endpoints and rules for Azure SQL Database and SQL Data Warehouse

10/22/2018 • 14 minutes to read • [Edit Online](#)

*Virtual network rules* are one firewall security feature that controls whether your Azure [SQL Database](#) or [SQL Data Warehouse](#) server 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.

## 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.

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](#).

## Terminology and description

**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.

## Benefits of a virtual network rule

Until you take action, the VMs on your subnets cannot communicate with your SQL Database. One action that establishes the communication is the creation of a virtual network rule. The rationale for choosing the VNet rule approach requires a compare-and-contrast discussion involving the competing security options offered by the firewall.

### A. Allow access to Azure services

The firewall pane has an **ON/OFF** button that is labeled **Allow access to Azure services**. The **ON** setting allows communications from all Azure IP addresses and all Azure subnets. These Azure IPs or subnets might not be owned by you. This **ON** setting is probably more open than you want your SQL Database to be. The virtual

network rule feature offers much finer granular control.

## B. IP rules

The SQL Database 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. But many nodes inside the Azure private network are configured with *dynamic* IP addresses. Dynamic IP addresses might change, such as when your VM is restarted. It would be folly to specify a dynamic IP address in a firewall rule, in a production environment.

You can salvage the IP option 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 to establish and to manage.

## C. 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.

However as of September 2017, the Azure SQL Database service is not yet among the services that can be assigned to a subnet.

# 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:

- A Web App can be mapped to a private IP in a VNet/subnet. Even if service endpoints are turned ON from the given VNet/subnet, connections from the Web App to the server will have an Azure public IP source, not a VNet/subnet source. To enable connectivity from a Web App to a server that has VNet firewall rules, you must **Allow Azure services to access server** on the server.
- 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 will fail.
  - The underlying reason is that MySQL and PostgreSQL do not presently support ACLing.
- 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 removing 'Allow Azure services to access server'

Many users want to remove **Allow Azure services to access server** from their Azure SQL Servers and replace it with a VNet Firewall Rule. However removing this affects the following Azure SQL Database features:

### Import Export Service

Azure SQL Database Import Export Service runs on VMs in Azure. These VMs are not in your VNet and hence get an Azure IP when connecting to your database. On removing **Allow Azure services to access server** these VMs will not be able to access your databases. You can work around the problem. Run the BACPAC import or export directly in your code by using the DACFx API. Ensure that this is deployed in a VM that is in the VNet-subnet for which you have set the firewall rule.

### SQL Database Query Editor

The Azure SQL Database Query Editor is deployed on VMs in Azure. These VMs are not in your VNet. Therefore the VMs get an Azure IP when connecting to your database. On removing **Allow Azure services to access server**, these VMs will not be able to access your databases.

### Table Auditing

At present there are two ways to enable auditing on your SQL Database. Table auditing fails after you have enabled service endpoints on your Azure SQL Server. Mitigation here is to move to Blob auditing.

### Impact on Data Sync

Azure SQL Database has the Data Sync feature that connects to your databases using Azure IPs. When using service endpoints, it is likely that you will turn off **Allow Azure services to access server** access to your logical server. This will break the Data Sync feature.

## Impact of using VNet Service Endpoints with Azure storage

Azure Storage has implemented the same feature that allows you to limit connectivity to your Storage account. If you choose to use this feature with a Storage account that is being used by an Azure SQL Server, you can run into issues. Next is a list and discussion of Azure SQL Database features that are impacted by this.

### Azure SQL Data Warehouse PolyBase

PolyBase is commonly used to load data into Azure SQL Data Warehouse from Storage accounts. If the 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. There is a mitigation for this, and you may contact Microsoft support for more information.

### 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 **IgnoreMissingServiceEndpoint** 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 VNs. You can use the **IgnoreMissingServiceEndpoint** flag to reduce or eliminate the downtime during transition.

You can set the **IgnoreMissingServiceEndpoint** 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.

A list of several SQL Database error messages is documented [here](#).

## 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 PowerShell script can also create virtual network rules. The crucial cmdlet **New-AzureRmSqlServerVirtualNetworkRule**. 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. Then navigate the portal to **SQL servers > Firewall / 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. 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

SQL server



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

ON

OFF

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.

Smiley face icon | Help icon | [REDACTED]@microsoft.co... MICROSOFT

## Create/Update virtual network rule

**\* Name** i

 ✓

provide vnet rule name

**Subscription** i

**\* Virtual network** i

 ▾

**\* Subnet name / Address prefix** i

 ▾

**OK**

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

10/23/2018 • 12 minutes to read • [Edit Online](#)

Both Azure [SQL Database](#) and [SQL Data Warehouse](#) support Virtual Service endpoints.

## 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.

This article does *not* apply to **Azure SQL Database Managed Instance** as you do not have a service endpoint associated with a Managed Instance subnet.

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](#).

## Major cmdlets

This article emphasizes the **New-AzureRmSqlServerVirtualNetworkRule** 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-AzureRmSqlServerVirtualNetworkRule**. In this article, these calls occur in [script 3 "Virtual network rule"](#):

1. [New-AzureRmVirtualNetworkSubnetConfig](#): Creates a subnet object.
2. [New-AzureRmVirtualNetwork](#): Creates your virtual network, giving it the subnet.
3. [Set-AzureRmVirtualNetworkSubnetConfig](#): Assigns a Virtual Service endpoint to your subnet.
4. [Set-AzureRmVirtualNetwork](#): Persists updates made to your virtual network.
5. [New-AzureRmSqlServerVirtualNetworkRule](#): 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-AzureRmAccount; }

#####
## Assignments to variables used by the later scripts. ##
#####

# You can edit these values, if necessary.

$SubscriptionName = 'yourSubscriptionName';
Select-AzureRmSubscription -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      = 'mysqldbserver-forvnet';
$SqlDbAdminLoginName   = 'ServerAdmin';
$SqlDbAdminLoginPassword = 'ChangeYourAdminPassword1';

$ServiceEndpointType_Name_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-AzureRmResourceGroup ` 
    -Name      $ResourceGroupName ` 
    -ErrorAction SilentlyContinue;

if ($null -eq $gottenResourceGroup)
{
    Write-Host "Creating your missing Resource Group - $ResourceGroupName.";

    $gottenResourceGroup = New-AzureRmResourceGroup ` 
        -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-AzureRmSqlServer ` 
    -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-AzureRmSqlServer ` 
    -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

```

#####
## 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-AzureRmVirtualNetworkSubnetConfig ` 
    -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-AzureRmVirtualNetwork ` 
    -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-AzureRmVirtualNetworkSubnetConfig ` 
    -Name        $SubnetName ` 
    -AddressPrefix $SubnetAddressPrefix ` 
    -VirtualNetwork $vnet ` 
    -ServiceEndpoint $ServiceEndpointTypeName_SQLDb;

Write-Host "Persist the updates made to the virtual network > subnet.";

$vnet = Set-AzureRmVirtualNetwork ` 
    -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-AzureRmVirtualNetwork ` 
    -ResourceGroupName $ResourceGroupName ` 
    -Name             $VNetName;

$subnet = Get-AzureRmVirtualNetworkSubnetConfig ` 
    -Name             $SubnetName ` 
    -VirtualNetwork $vnet;

Write-Host "Add the subnet .Id as a rule, into the ACLs for your Azure SQL Database server./";

$vnetRuleObject1 = New-AzureRmSqlServerVirtualNetworkRule ` 
    -ResourceGroupName     $ResourceGroupName ` 
    -ServerName           $SqlDbServerName ` 
    -VirtualNetworkRuleName $VNetRuleName ` 
    -VirtualNetworkSubnetId $subnet.Id;

$vnetRuleObject1;

Write-Host "Verify that the rule is in the SQL DB ACL./";

$vnetRuleObject2 = Get-AzureRmSqlServerVirtualNetworkRule ` 
    -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-AzureRmSqlServerVirtualNetworkRule ` 
    -ResourceGroupName $ResourceGroupName ` 
    -ServerName $SqlDbServerName ` 
    -VirtualNetworkRuleName $VNetRuleName ` 
    -ErrorAction SilentlyContinue;

Write-Host "Delete the endpoint from the subnet.";

$vnet = Get-AzureRmVirtualNetwork ` 
    -ResourceGroupName $ResourceGroupName ` 
    -Name $VNetName;

Remove-AzureRmVirtualNetworkSubnetConfig ` 
    -Name $SubnetName ` 
    -VirtualNetwork $vnet;

Write-Host "Delete the virtual network (thus also deletes the subnet).";

Remove-AzureRmVirtualNetwork ` 
    -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-AzureRmSqlServer ` 
        -ServerName $SqlDbServerName ` 
        -ResourceGroupName $ResourceGroupName ` 
        -ErrorAction SilentlyContinue;

    Write-Host "Remove the Azure Resource Group.';

    Remove-AzureRmResourceGroup ` 
        -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".';

```

## Actual output from scripts 1 through 4

The output from our test run is displayed next, in an abbreviated format. The output might be helpful in case you do not want to actually run the PowerShell scripts now.

```
[C:\WINDOWS\system32\]
0 >> C:\Demo\PowerShell\sql-database-vnet-service-endpoint-powershell-s1-variables.ps1
Do you need to log into Azure (only one time per powershell.exe session)? [yes/no]: yes
```

```
Environment      : AzureCloud
Account         : xx@microsoft.com
TenantId        : 11111111-1111-1111-1111-111111111111
SubscriptionId  : 22222222-2222-2222-2222-222222222222
SubscriptionName : MySubscriptionName
CurrentStorageAccount :
```

```
[C:\WINDOWS\system32\]
0 >> C:\Demo\PowerShell\sql-database-vnet-service-endpoint-powershell-s2-prerequisites.ps1
Check whether your Resource Group already exists.
Creating your missing Resource Group - RG-YourNameHere.
```

```
ResourceGroupName : RG-YourNameHere
Location         : westcentralus
ProvisioningState : Succeeded
Tags             :
ResourceId       : /subscriptions/22222222-2222-2222-2222-222222222222/resourceGroups/RG-YourNameHere
```

```
Check whether your Azure SQL Database server already exists.
Creating the missing Azure SQL Database server - mysqlserver-forvnet.
Gather the credentials necessary to next create an Azure SQL Database server.
Create your Azure SQL Database server.
```

```
ResourceGroupName      : RG-YourNameHere
ServerName            : mysqlserver-forvnet
Location              : westcentralus
SqlAdministratorLogin : ServerAdmin
SqlAdministratorPassword :
ServerVersion         : 12.0
Tags                 :
Identity              :
```

```
Completed script 2, the "Prerequisites".
```

```
[C:\WINDOWS\system32\]
0 >> C:\Demo\PowerShell\sql-database-vnet-service-endpoint-powershell-s3-vnet-rule.ps1
Define a subnet 'mySubnet', to be given soon to a virtual network.
Create a virtual network 'myVNet'. Give the subnet to the virtual network that we created.
WARNING: The output object type of this cmdlet will be modified in a future release.
Assign a Virtual Service endpoint 'Microsoft.Sql' to the subnet.
Persist the updates made to the virtual network > subnet.
```

```
Get the subnet object.
Add the subnet .Id as a rule, into the ACLs for your Azure SQL Database server.
ProvisioningState Service      Locations
----- -----
Succeeded      Microsoft.Sql {westcentralus}
```

```
Verify that the rule is in the SQL DB ACL.
```

```
Completed script 3, the "Virtual-Network-Rule".
```

```
[C:\WINDOWS\system32\]
0 >> C:\Demo\PowerShell\sql-database-vnet-service-endpoint-powershell-s4-clean-up.ps1
```

```
Delete the rule from the SQL DB ACL.
```

```
Delete the endpoint from the subnet.
```

```
Delete the virtual network (thus also deletes the subnet).
```

```
CAUTION !: Do you want to DELETE your Azure SQL Database server AND your Resource Group? [yes/no]: yes  
Remove the Azure SQL DB server.
```

```
ResourceGroupName      : RG-YourNameHere  
ServerName            : mysqlserver-forvnet  
Location              : westcentralus  
SqlAdministratorLogin : ServerAdmin  
SqlAdministratorPassword :  
ServerVersion         : 12.0  
Tags                  :  
Identity              :
```

```
Remove the Azure Resource Group.
```

```
True
```

```
Completed script 4, the "Clean-Up".
```

This is the end of our main PowerShell script.

## 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-AzureRmAccount; }  
  
# Assignments to variables used by the later scripts.
```

```

# You can EDIT these values, if necessary.

$SubscriptionName = 'yourSubscriptionName';
Select-AzureRmSubscription -SubscriptionName "$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-AzureRmVirtualNetwork `

-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-AzureRmVirtualNetworkSubnetConfig ` 
    -Name          $SubnetName ` 
    -AddressPrefix $SubnetAddressPrefix ` 
    -VirtualNetwork $vnet ` 
    -ServiceEndpoint $ServiceEndpointTypeName_SQLDb;

# Persist the subnet update.
$vnet = Set-AzureRmVirtualNetwork ` 
    -VirtualNetwork $vnet;

for ($nn=0; $nn -lt $vnet.Subnets.Count; $nn++)
{ $vnet.Subnets[0].ServiceEndpoints; } # Display.

```

## Actual output

The following block displays our actual feedback (with cosmetic edits).

```

<# Our output example (with cosmetic edits), when the subnet was already tagged:

Do you need to log into Azure (only one time per powershell.exe session)? [yes/no]: no

Environment      : AzureCloud
Account         : xx@microsoft.com
TenantId        : 11111111-1111-1111-1111-111111111111
SubscriptionId   : 22222222-2222-2222-2222-222222222222
SubscriptionName : MySubscriptionName
CurrentStorageAccount :

ProvisioningState : Succeeded
Service          : Microsoft.Sql
Locations        : {westcentralus}

Good: Subnet found, and is already tagged as an endpoint of type 'Microsoft.Sql'.
#>

```

# Azure SQL Database and SQL Data Warehouse access control

10/16/2018 • 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 logical 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. Use Active Directory authentication (integrated security) [whenever possible](#). 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

9/25/2018 • 11 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](#).

## 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 of your SQL server.

The screenshot shows the Azure portal interface for managing SQL servers. On the left, there's a sidebar with icons for creating new resources. The main area shows a list of subscriptions under 'Subscriptions: Test Subscriber'. Two items are listed: 'TestSQLServer' and 'Test2'. A red box highlights 'TestSQLServer'. To the right, the properties for 'TestSQLServer' are displayed. The 'Overview' section is highlighted with a red box. The 'Essentials' pane on the right also has a red box around it. The 'Essentials' pane displays the following information:

- Resource group: Group1
- Server version: V12
- Status: Available
- Location: West US 2
- Subscription name: SQL DB Content
- Subscription ID: 45651c6...
- Server admin: Rick
- Active Directory admin: rickb@contoso.com
- Firewall: Show firewall settings

### • 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.

### • 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:

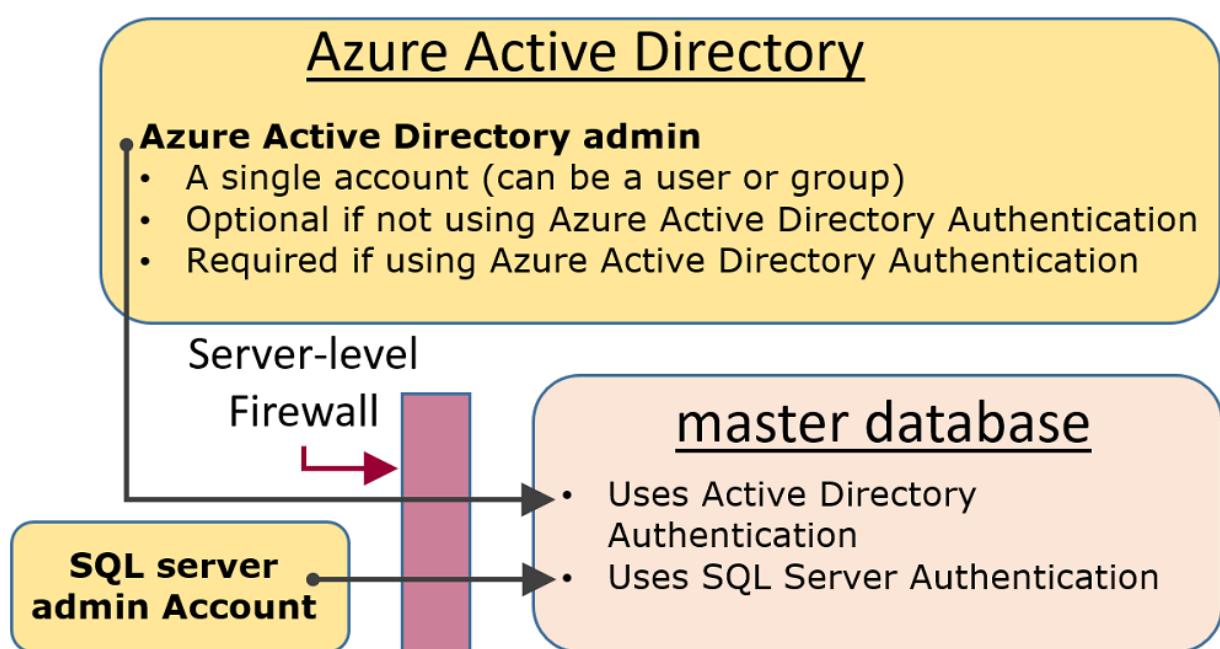
- Are the only accounts that can automatically connect to any SQL Database on the server. (To connect to a user database, other accounts must either be the owner of the database, or have a user account in the user database.)
- These accounts enter user databases as the `dbo` user and they have all the 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 firewall rules.
- Can add and remove members to the `dbmanager` and `loginmanager` roles.
- Can view the `sys.sql_logins` system table.

## 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 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.



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 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

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. Optional step: 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 by 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 using the **ALTER ROLE** statement.

```
ALTER ROLE db_owner ADD MEMBER Mary;
```

### NOTE

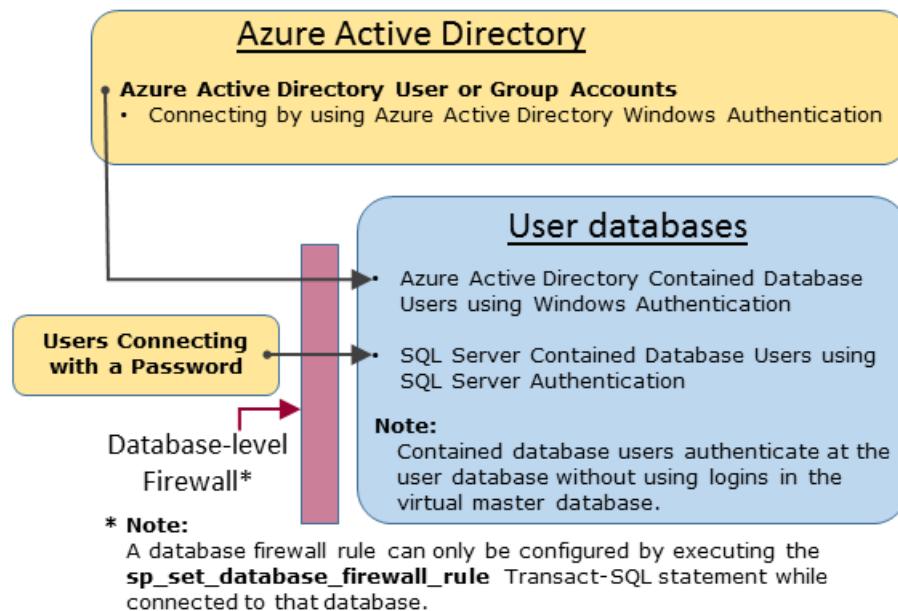
One common reason to create a database user based on a logical server login is for users that need access to multiple databases. Since contained database users are individual entities, each database maintains its own user and its own password. This can cause overhead as the user must then remember each 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
```

- 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

10/8/2018 • 8 minutes to read • [Edit Online](#)

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).

## 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](#).

## 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.

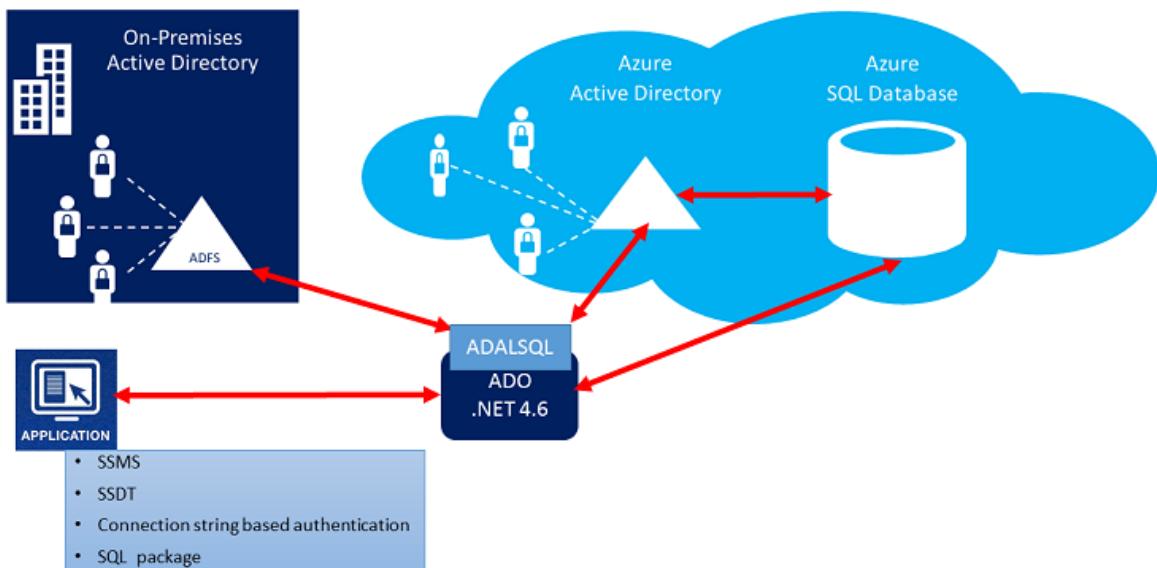
**NOTE**

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](#).

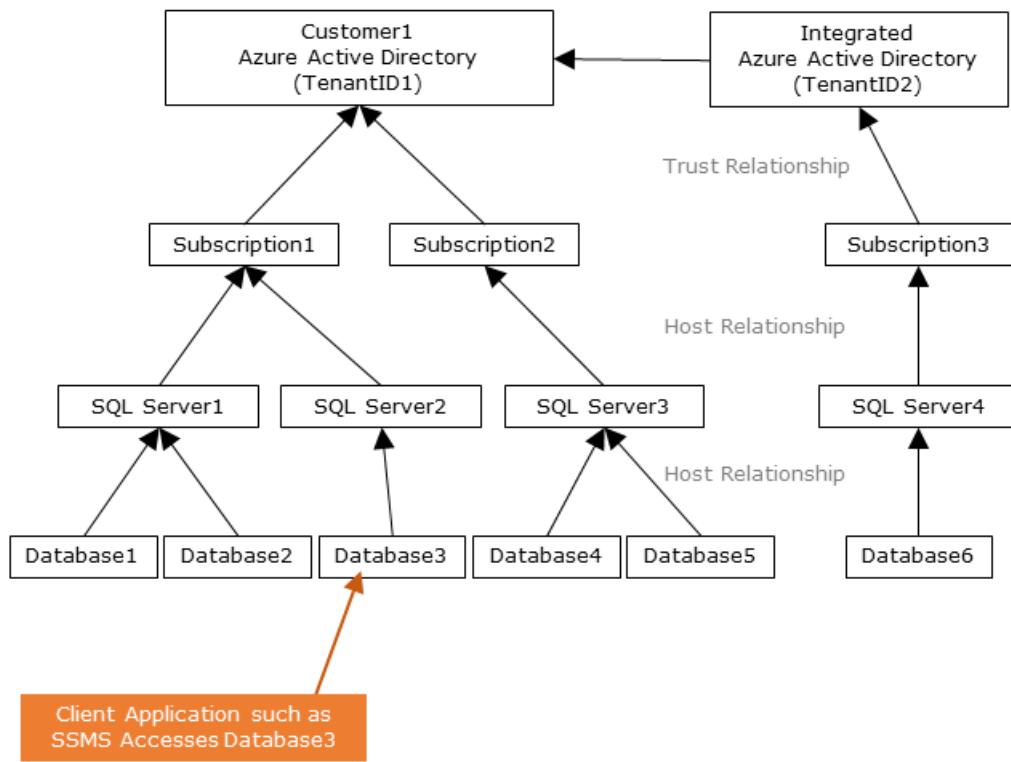
## Trust architecture

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

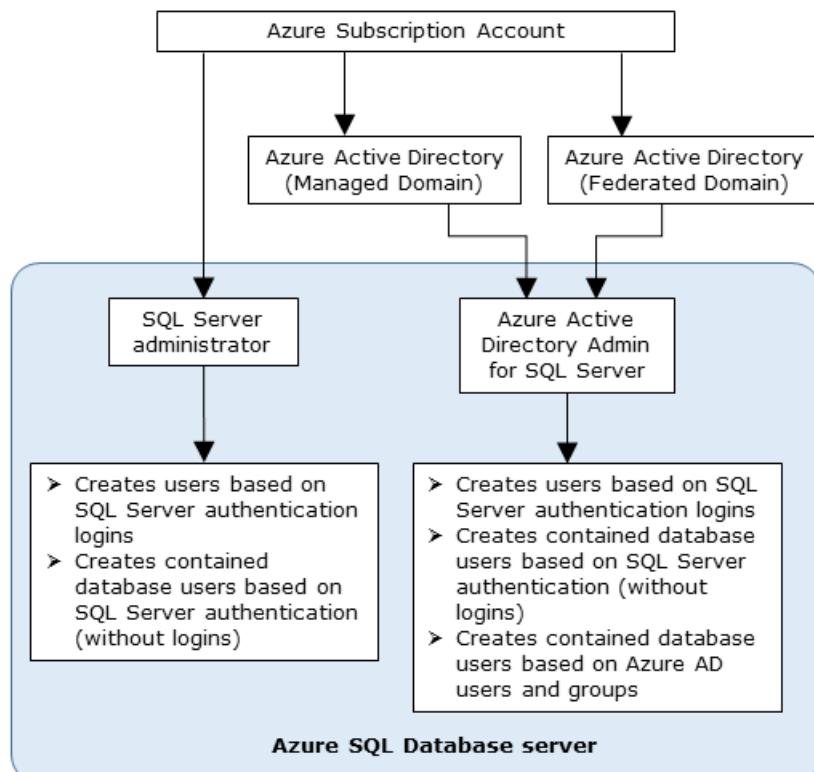


The following diagram indicates the federation, trust, and hosting relationships that allow a client to connect to a database by submitting a token. The token is authenticated by an Azure AD, and is trusted by the database. Customer 1 can represent an Azure Active Directory with native users or an Azure 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 limitations related to Managed Instance:

- Only Azure AD admin can create databases, Azure AD users are scoped to a single DB and do not have this permission
- Database ownership:
  - Azure AD principal cannot change ownership of the database (`ALTER AUTHORIZATION ON DATABASE`) and cannot be set as owner.
  - For databases created by Azure AD admin no ownership is set (`owner_sid` field in `sys.sysdatabases` is `0x1`).
- SQL Agent cannot be managed when logged in using Azure AD principals.
- Azure AD admin cannot be impersonated using `EXECUTE AS`
- DAC connection is not supported with Azure AD principals.

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>)`

## Connecting using Azure AD identities

Azure Active Directory authentication supports the following methods of connecting to a database using Azure AD identities:

- Using integrated Windows authentication
- Using an Azure AD principal name and a password
- Using Application token authentication

## 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, Managed Instance, or Azure SQL Data Warehouse at any time.
- 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.
- **ODBC version 13.1** supports Azure Active Directory authentication however `bcp.exe` cannot connect using Azure Active Directory authentication because it uses an older ODBC provider.
- `sqlcmd` supports Azure Active Directory authentication beginning with version 13.1 available from the [Download Center](#).
- 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 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](#).

# Configure and manage Azure Active Directory authentication with SQL

10/8/2018 • 19 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](#) and [SQL Data Warehouse](#). For an overview, see [Azure Active Directory Authentication](#).

## 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.

## 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.

**Additional information:** 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 in Azure AD. Following steps describe the process of granting permissions for users with different privileges in directory.

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.

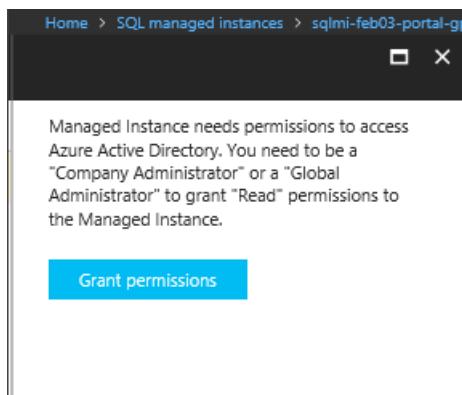
1. In the Azure portal, in the upper-right corner, select your connection to drop down a list of possible Active Directories.
2. Choose the correct Active Directory as the default Azure AD.

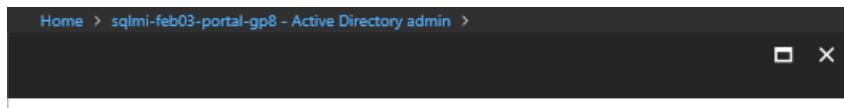
This step links the subscription associated with Active Directory with Managed Instance making sure that the same subscription is used for both Azure AD and the Managed Instance.

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, there's a list of instances including 'mi-workshop2'. The 'mi-workshop2' row is highlighted with a red box. On the right, the 'Active Directory admin' section is displayed, also with a red box around its title. The page includes a warning banner about granting permissions to access Azure Active Directory.

4. Select on banner on top of Active Directory admin page. If you are logged in as Global/Company administrator in Azure AD, you can do it from Azure portal or using PowerShell.





```
# Gives Azure Active Directory read permission to a Service Principal representin
# Can be executed only by a "Company Administrator" or "Global Administrator" typ
#
$aadTenant = "0c1edf5d-e5c5-4aca-ab69-ef194134f44b"
$principalId = "0e696736-4ba3-437f-88a1-f8573bf34e17"
# Connect to AAD directory
$adminUser = Connect-AzureAD -AzureEnvironmentName AzureCloud -TenantId $aadTenant
# Get Azure AD role "Directory Readers" 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 "Directory Readers"} | Enable-AzureADDirectoryRole -RoleTemplateId $roleTemplate.ObjectId
    $role = Get-AzureADDirectoryRole | Where-Object {$_ . displayName -eq $roleName}
}
# Get service principal for managed instance
$roleMember = Get-AzureADServicePrincipal -ObjectId $principalId
$roleMember.Count
if ($roleMember -eq $null)
{
    Write-Output "Error: No Service Principals with ID '$($principalId)'."
    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 $null)
{
    # Add principal to readers role
    Write-Output "Adding service principal '$($principalId)' to 'Directory Reader' role"
    Add-AzureADDirectoryRoleMember -ObjectId $role.ObjectId -RefObjectId $roleMember.ObjectId
    Write-Output "'$($principalId)' service principal added to 'Directory Readers' role"
}
else
{
    Write-Output "Service principal '$($principalId)' is already member of 'Directory Readers' role"
}
```

If you are logged in as Global/Company administrator in Azure AD, you can do it from the Azure portal or execute a PowerShell script.

- After operation is successfully completed, following notification will show up in top right corner:



- 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. On the left, there's a sidebar with 'SQL managed instances' and a list containing 'sqlmi-feb03-portal'. A red box highlights the list item 'sqlmi-feb03-portal-gp8'. On the right, the main content area is titled 'sqlmi-feb03-portal-gp8 - Active Directory admin'. It features a search bar at the top. Below it is a navigation menu with options like 'Overview', 'Activity log', 'Tags', 'Diagnose and solve problems', 'Connection strings', and 'Active Directory admin'. The 'Active Directory admin' option is highlighted with a red box. Further down are sections for 'Pricing tier', 'Locks', and 'Automation script'. At the bottom, there are links for 'Resource health' and 'New support request'. A status bar at the bottom indicates 'Active Directory admin' and 'No Active Directory admin'.

7. 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 [Azure AD Features and Limitations](#). Role-based access control (RBAC) applies only to the Azure portal and is not propagated to SQL Server.

The screenshot shows the 'Add admin' dialog box. At the top, it says 'Add admin' and 'Active Directory admin'. Below that is a 'Select' button with a plus sign and the word 'Invite'. A search bar contains the text 'rick b'. A list of users follows, with 'Rick B rickb@contoso.com' highlighted with a red box. A red arrow points from this highlighted user to a red box around the 'Select' button at the bottom of the dialog. To the left of the dialog is a vertical sidebar with icons for 'SQL servers', 'byham - Active Directory admin', and 'Add admin'.

8. At the top of the Active Directory admin page, select **Save**.

 Set admin

 Remove admin

 Save



Azure Active Directory authentication allows you to connect to your Azure SQL Database V12.

[Learn more](#) 

Active Directory admin 

 rickb@contoso.com

The process of changing the administrator may take several minutes. Then the new administrator appears in the Active Directory admin box.

#### IMPORTANT

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 and rolling back its creation, 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.

#### TIP

To later remove an Admin, at the top of the Active Directory admin page, select **Remove admin**, and then select **Save**.

## Provision an Azure Active Directory administrator for your Azure SQL Database server

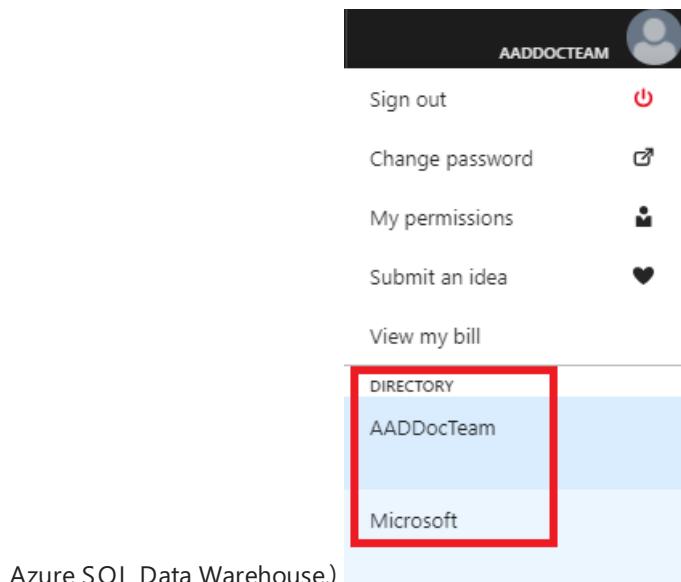
#### IMPORTANT

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



2. In the left banner select **All services**, and in the filter type in **SQL server**. Select **Sql Servers**.

Microsoft Azure

1 + Create a resource

2 All services

3 sql server

All services

FAVORITES

Dashboard

Resource groups

Azure Database for MySQL servers

SQL data warehouses  
Keywords: SQL Server

SQL servers

Azure Database for PostgreSQL servers

SQL databases  
Keywords: SQL Server

Virtual machines  
Keywords: 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 Microsoft Azure portal interface. On the left, there's a sidebar with various icons. The main area is titled "SQL servers" under "test\_srv · Active Directory admin".  
In the center-left pane, there's a table with one item: "test\_srv" (with a "sql\_v12" icon). A red box highlights this row, and a red arrow points from it to the "Active Directory admin" section in the center-right pane.  
The right-hand pane is titled "test\_srv - Active Directory admin" and contains the following sections:

- Actions:** Set admin, Remove admin, Save
- Information:** Azure Active Directory authentication all, Azure SQL Database V12, Learn more
- Status:** Active Directory admin (No Active Directory admin)
- Settings:** 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 (highlighted with a red box), Deleted databases, Properties

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 box within the Microsoft Azure interface. At the top, there's a 'Select' button with a plus sign and the word 'Invite'. Below it is a search bar containing 'rick b' with a green checkmark. A list of users follows, with 'Rick B' (rickb@contoso.com) highlighted with a red box and a red arrow pointing down to the 'Select' button at the bottom. Other users listed are 'Rick F' (rickf@live.com), 'Rick G' (rickg@contoso.com), and 'Rick M' (rickm@northwind.co). The 'Selected' section at the bottom contains 'Rick B' and a lock icon. The 'Select' button at the bottom 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. At the top, there are three buttons: 'Set admin' (highlighted with a red box), 'Remove admin', and 'Save'. Below this, there's a section about Azure Active Directory authentication. Further down, the 'Active Directory admin' section shows the email 'rickb@contoso.com' with a pencil icon for editing.

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

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-AzureRmAccount
- Select-AzureRmSubscription

Cmdlets used to provision and manage Azure AD admin:

| CMDLET NAME   | DESCRIPTION   |
|---|---|
| Set-AzureRmSqlServerActiveDirectoryAdministrator    | Provisions an Azure Active Directory administrator for Azure SQL server or Azure SQL Data Warehouse. (Must be from the current subscription.) |
| Remove-AzureRmSqlServerActiveDirectoryAdministrator | Removes an Azure Active Directory administrator for Azure SQL server or Azure SQL Data Warehouse.   |
| Get-AzureRmSqlServerActiveDirectoryAdministrator    | 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-AzureRmSqlServerActiveDirectoryAdministrator .
```

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-AzureRmSqlServerActiveDirectoryAdministrator -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-AzureRmSqlServerActiveDirectoryAdministrator` 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-AzureRmSqlServerActiveDirectoryAdministrator -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-AzureRmSqlServerActiveDirectoryAdministrator -ResourceGroupName "Group-23" -ServerName "demo_server" |
Format-List
```

The following example removes an Azure AD administrator:

```
Remove-AzureRmSqlServerActiveDirectoryAdministrator -ResourceGroupName "Group-23" -ServerName "demo_server"
```

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](#)

#### CLI

You can also provision an Azure AD admin by calling the following CLI commands:

| COMMAND                                       | DESCRIPTION   |
|---|---|
| <a href="#">az sql server ad-admin create</a> | Provisions an Azure Active Directory administrator for Azure SQL server or Azure SQL Data Warehouse. (Must be from the current subscription.) |
| <a href="#">az sql server ad-admin delete</a> | Removes an Azure Active Directory administrator for Azure SQL server or Azure SQL Data Warehouse.   |
| <a href="#">az sql server ad-admin list</a>   | Returns information about an Azure Active Directory administrator currently configured for the Azure SQL server or Azure SQL Data Warehouse.  |
| <a href="#">az sql server ad-admin update</a> | Updates the Active Directory administrator for an Azure SQL server or Azure SQL Data Warehouse.   |

For more information about CLI commands, see [SQL - az sql](#).

## 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 (**ADALSQL.DLL**) is available in multiple languages (both x86 and amd64) from the download center at [Microsoft Active Directory Authentication Library for Microsoft SQL Server](#).

You can meet these requirements by:

- Installing either [SQL Server 2016 Management Studio](#) or [SQL Server Data Tools for Visual Studio 2015](#) meets the .NET Framework 4.6 requirement.
- SSMS installs the x86 version of **ADALSQL.DLL**.
- SSDT installs the amd64 version of **ADALSQL.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 **ADALSQL.DLL**.

## Create contained database users in your database mapped to Azure AD identities

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.

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;
```

**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.

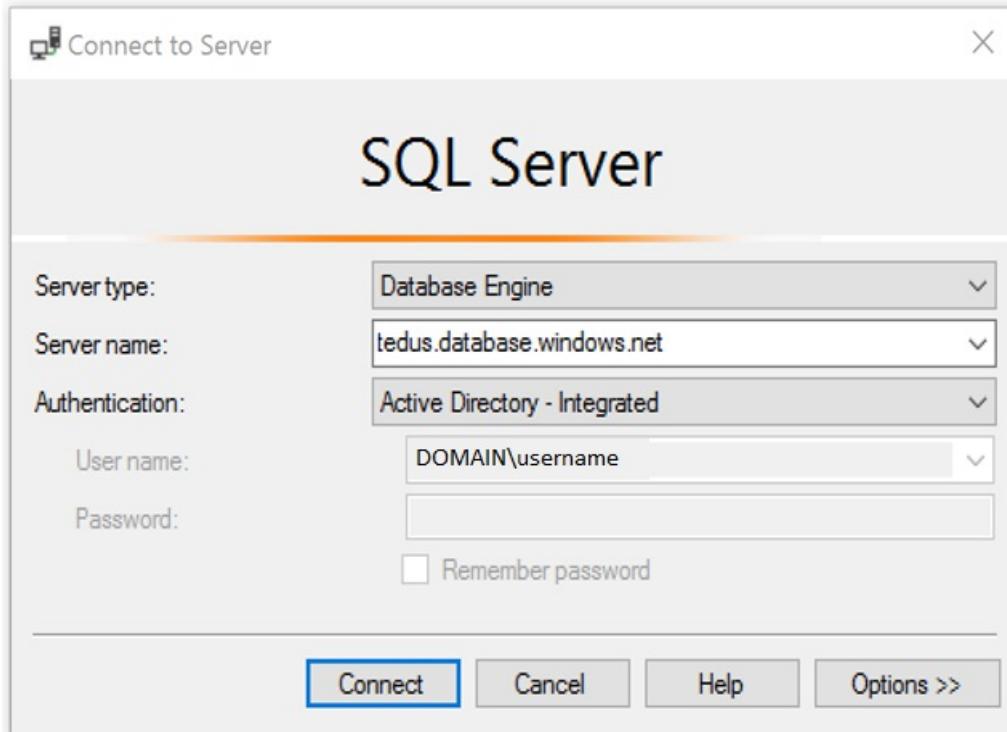
# Using an Azure AD identity to connect using SSMS or SSDT

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.

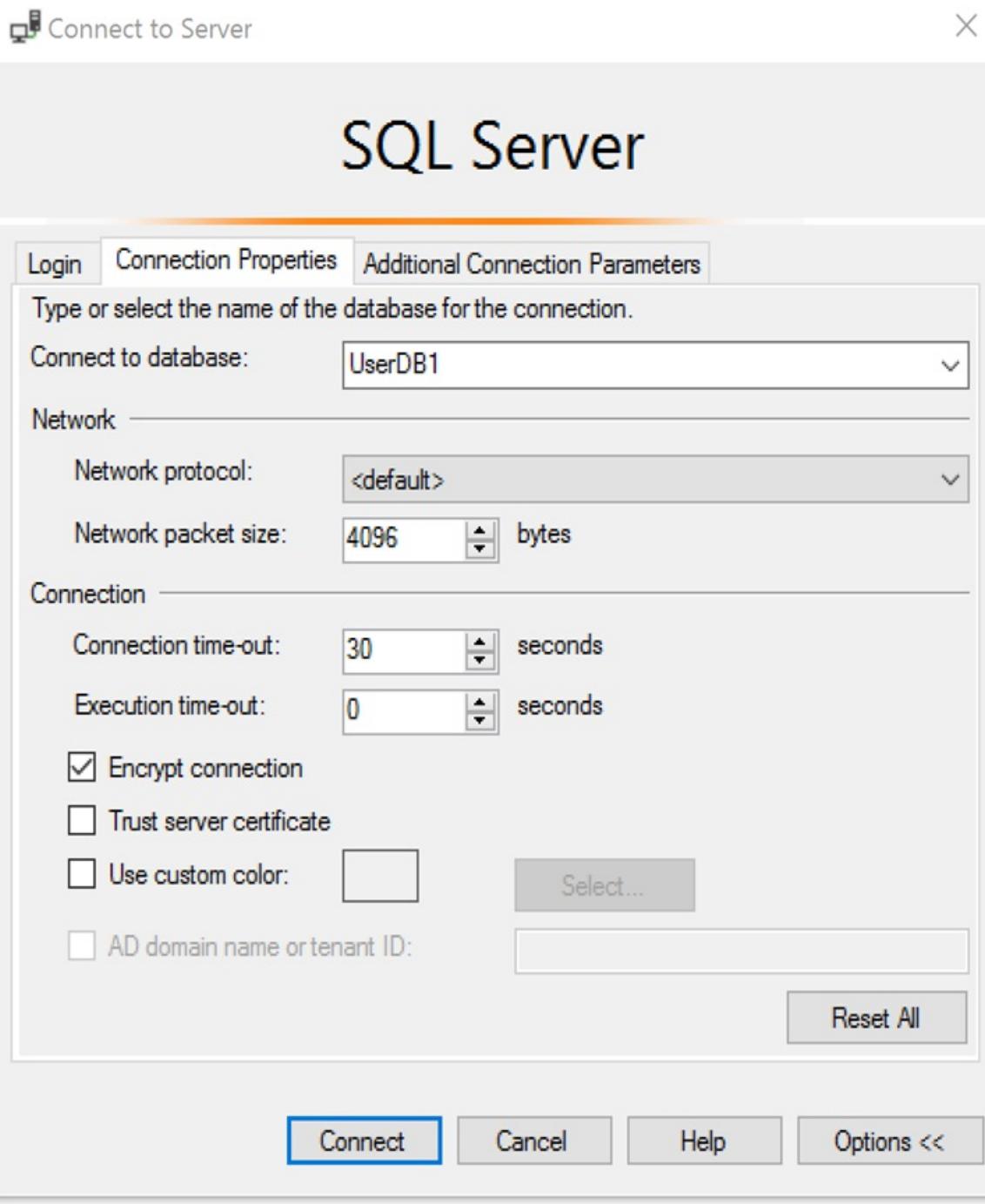
## Active Directory integrated authentication

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.)



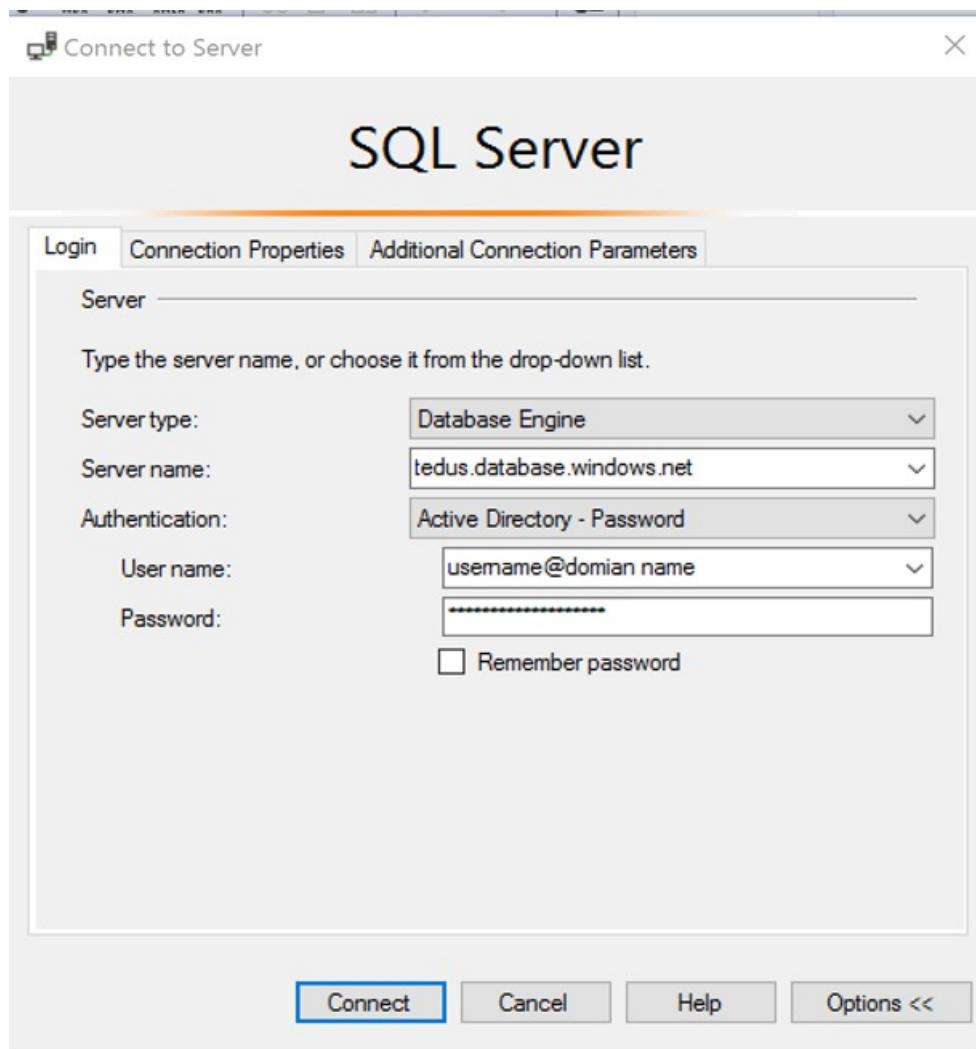
## Active Directory password authentication

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 his windows credential, but his local machine is not joined with the domain ( i.e. using a remote access). In this case a Windows user can indicate his 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**. This 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);
connection.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](#).

```
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
```

## 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](#).

# Conditional Access (MFA) with Azure SQL Database and Data Warehouse

9/24/2018 • 2 minutes to read • [Edit Online](#)

Both Azure [SQL Database](#) 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 a 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

+ Overview Quick start

MANAGE

- Users and groups
- Enterprise applications
- App registrations
- Application proxy
- Licenses
- Azure AD Connect
- Domain names
- Mobility (MDM and MAM)
- Password reset
- Company branding
- User settings
- Properties
- Notifications settings

SECURITY

- Conditional access
- Users flagged for risk

Classic portal Switch directory Delete directory

Users and groups

Enterprise applications

USERS SIGN-INS

Sync with Windows Server AD

Sync users and groups from your on-premises directory to your Azure AD

Self-service password reset

Enable your users to reset their forgotten passwords

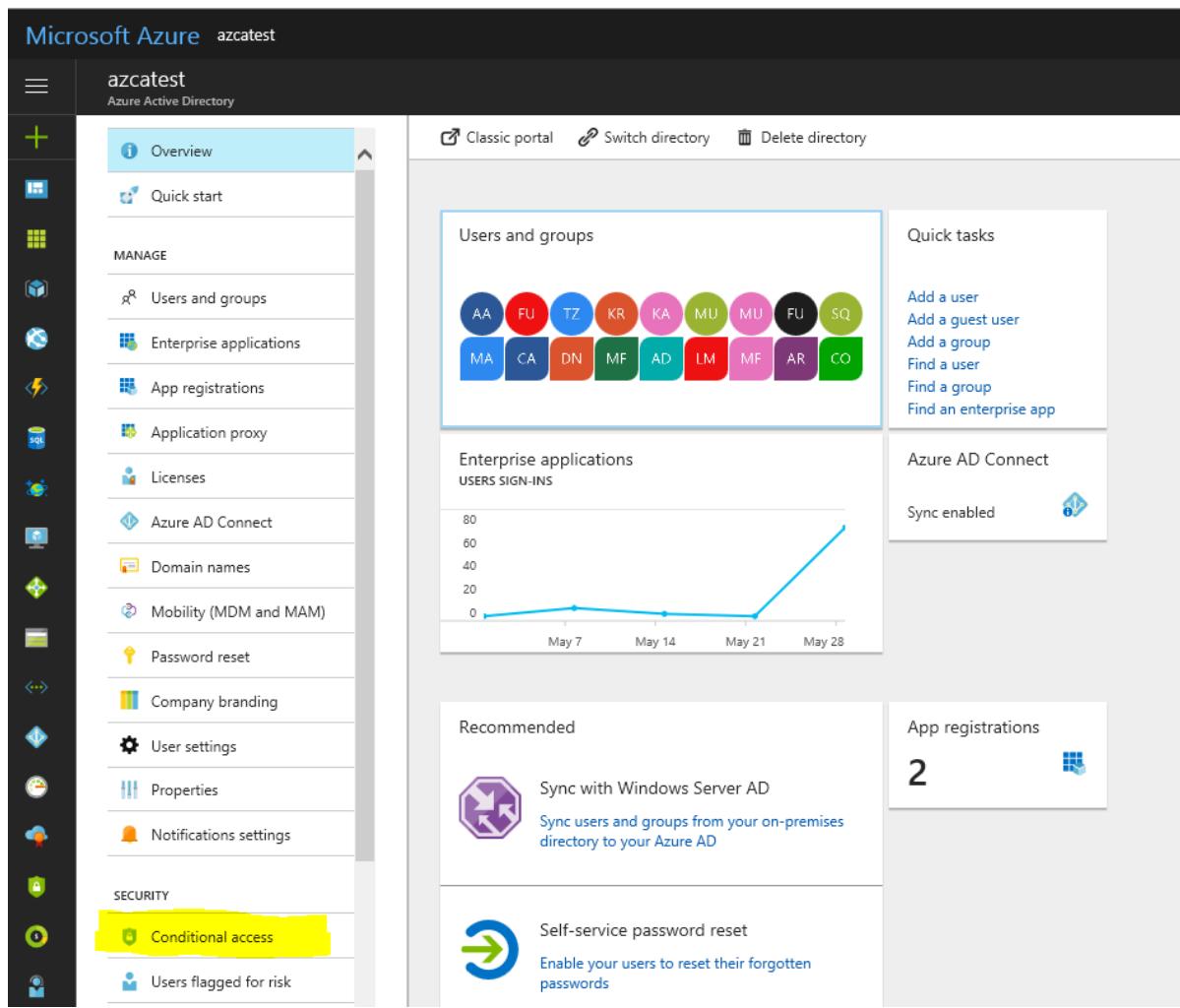
Quick tasks

- Add a user
- Add a guest user
- Add a group
- Find a user
- Find a group
- Find an enterprise app

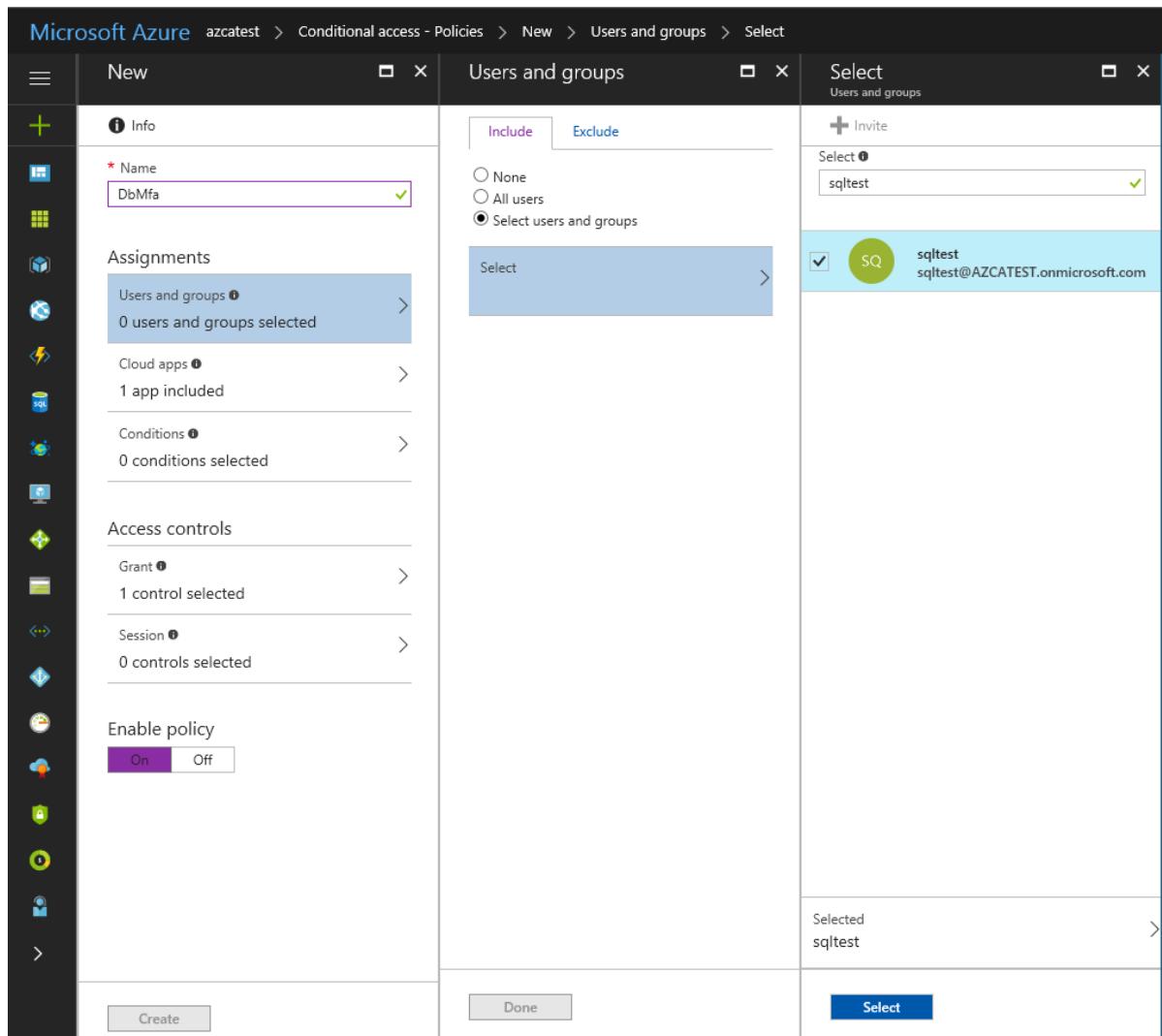
Azure AD Connect

Sync enabled

App registrations 2



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'. In the 'Assignments' section, 'Cloud apps' is selected, showing '1 app included' which is 'Azure SQL Database'. The 'Access controls' section shows 'Grant' selected with '1 control selected'. The 'Enable policy' section has 'On' selected. At the bottom right is a 'Done' button.

If you can't find **Azure SQL Database** listed in the following third screen shot, 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. The left sidebar lists various policy types with icons. The main area has two tabs: 'New' (selected) and 'Grant'. The 'Info' section contains a name field ('DbMfa') with a checkmark. The 'Assignments' section includes 'Users and groups', 'Cloud apps', and 'Conditions' sections. The 'Access controls' section shows 'Grant' selected with '1 control selected'. The 'Enable policy' section has an 'On' button highlighted. The 'Grant' tab is active, showing options to 'Select the controls to be enforced'. It includes radio buttons for 'Block access' (unselected) and 'Grant access' (selected), and checkboxes for 'Require multi-factor authentication' (selected), 'Require device to be marked as compliant' (unchecked), and 'Require domain joined device' (unchecked). A 'For multiple controls' section at the bottom has a radio button for 'Require all the selected controls' (selected) and another for 'Require one of the selected controls (preview)' (unchecked). Buttons for 'Create' and 'Select' are at the bottom.

## 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.

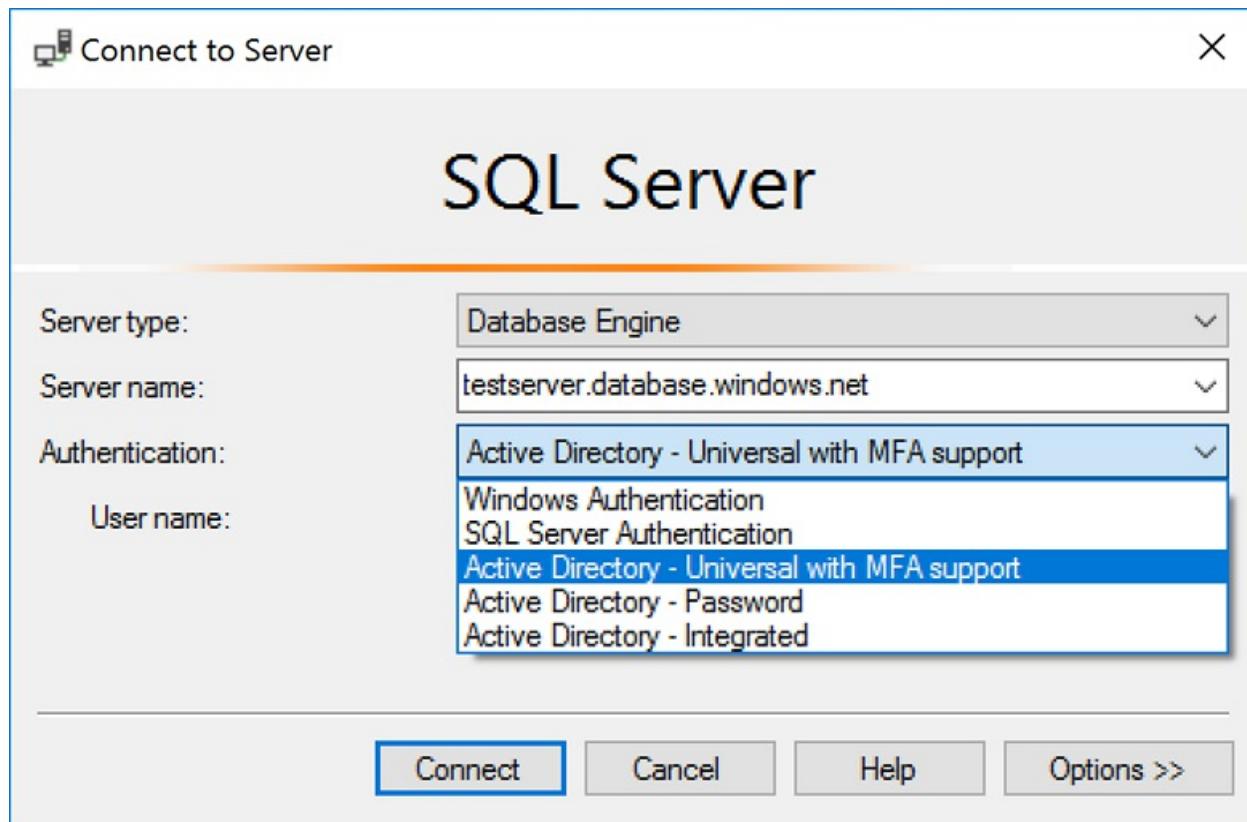
## Next steps

For a tutorial, see [Secure your Azure SQL Database](#).

# Universal Authentication with SQL Database and SQL Data Warehouse (SSMS support for MFA)

10/8/2018 • 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*. **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 in this article, use at least July 2017, version 17.2. The most recent connection dialog box, looks like this:



## The five authentication options

- Active Directory Universal Authentication supports the two non-interactive authentication methods (`Active Directory - Password` authentication and `Active Directory - Integrated` authentication). Non-interactive `Active Directory - Password` and `Active Directory - Integrated` Authentication methods can be used in many different applications (ADO.NET, JDBC, ODBC, etc.). These two methods never result in pop-up dialog boxes.
- `Active Directory - Universal with MFA` authentication is an interactive method that also supports *Azure Multi-Factor Authentication* (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.

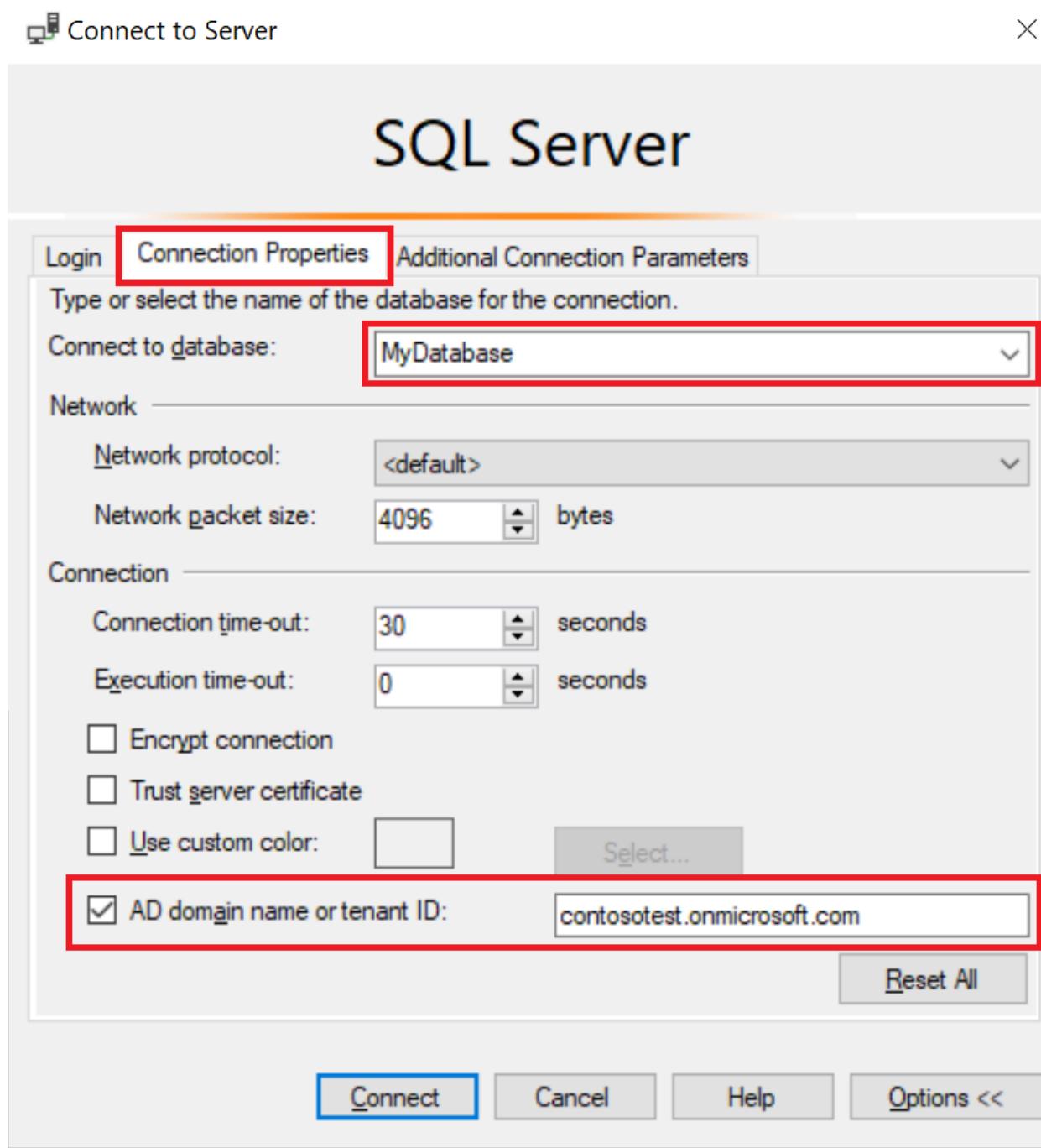
For a description of Multi-Factor Authentication, see [Multi-Factor Authentication](#). For configuration steps, see [Configure Azure SQL Database multi-factor authentication for SQL Server Management Studio](#).

## 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.



### 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 (i.e. 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

9/25/2018 • 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.

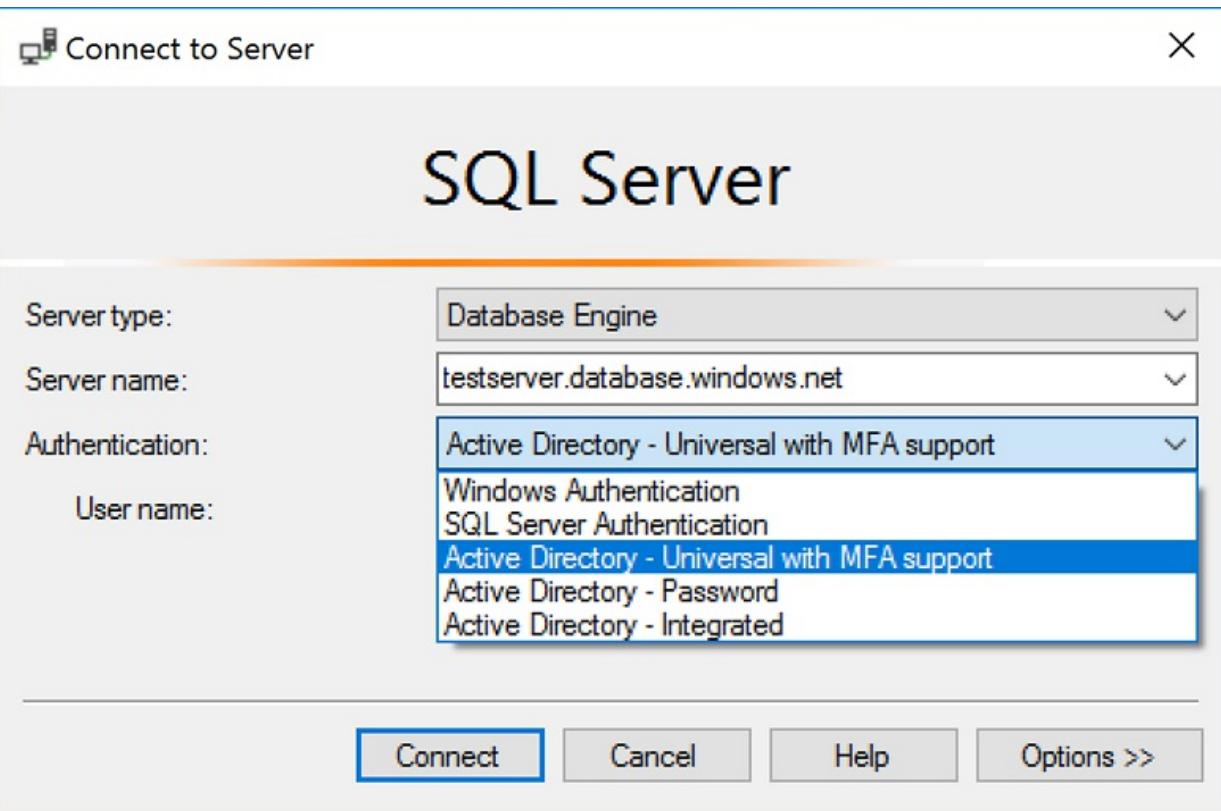
## Configuration steps

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.

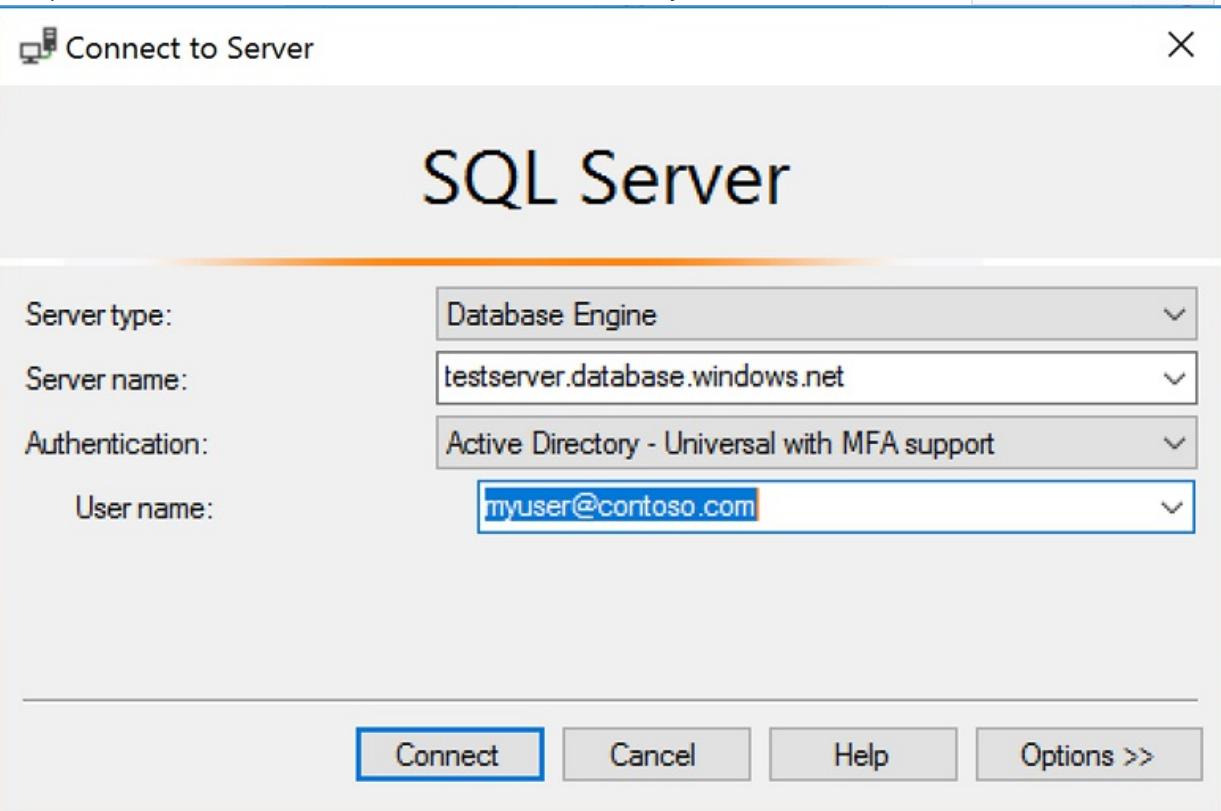
## Connecting by using universal authentication with SSMS

The following steps show how to connect to SQL Database or SQL Data Warehouse by using the latest SSMS.

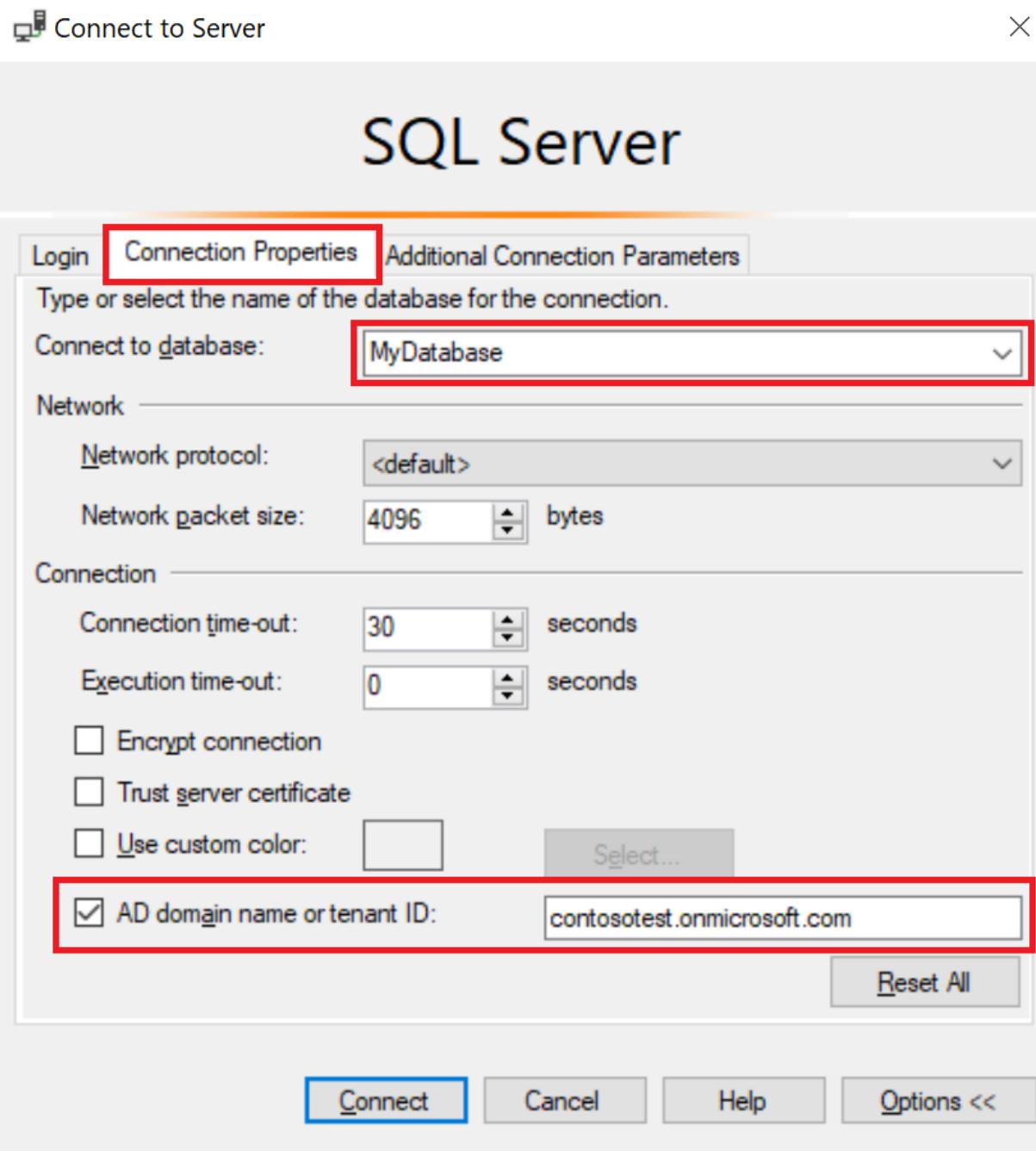
1. To connect using Universal Authentication, on the **Connect to Server** dialog box, select **Active Directory - Universal with MFA support**. (If you see **Active Directory Universal Authentication** you are not on the latest version of SSMS.)



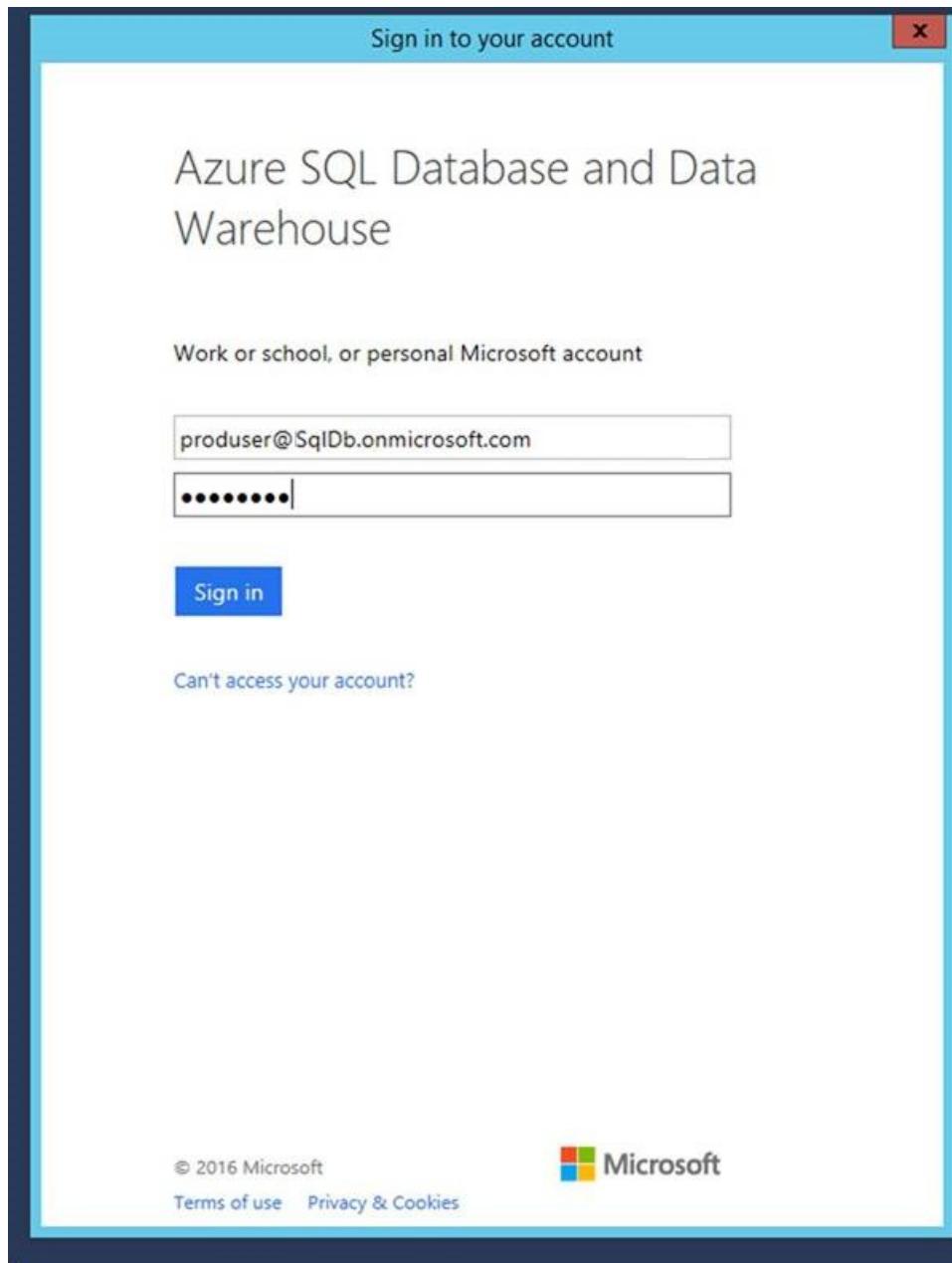
2. Complete the **User name** box with the Azure Active Directory credentials, in the format `user_name@domain.com`.



3. If you are connecting as a guest user, you must click **Options**, and on the **Connection Property** dialog box, complete the **AD domain name or tenant ID** box. For more information, see [Universal Authentication with SQL Database and SQL Data Warehouse \(SSMS support for MFA\)](#).



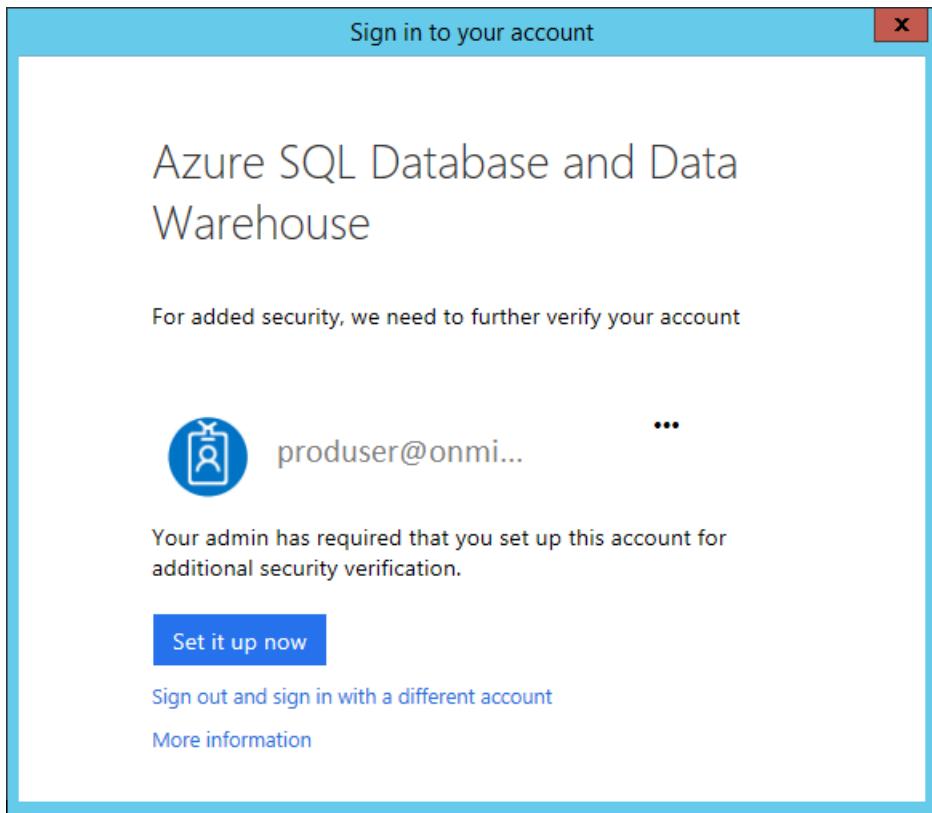
4. As usual for SQL Database and SQL Data Warehouse, you must click **Options** and specify the database on the **Options** dialog box. (If the connected user is a guest user ( i.e. joe@outlook.com), you must check the box and add the current AD domain name or tenant ID as part of Options. See [Universal 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:

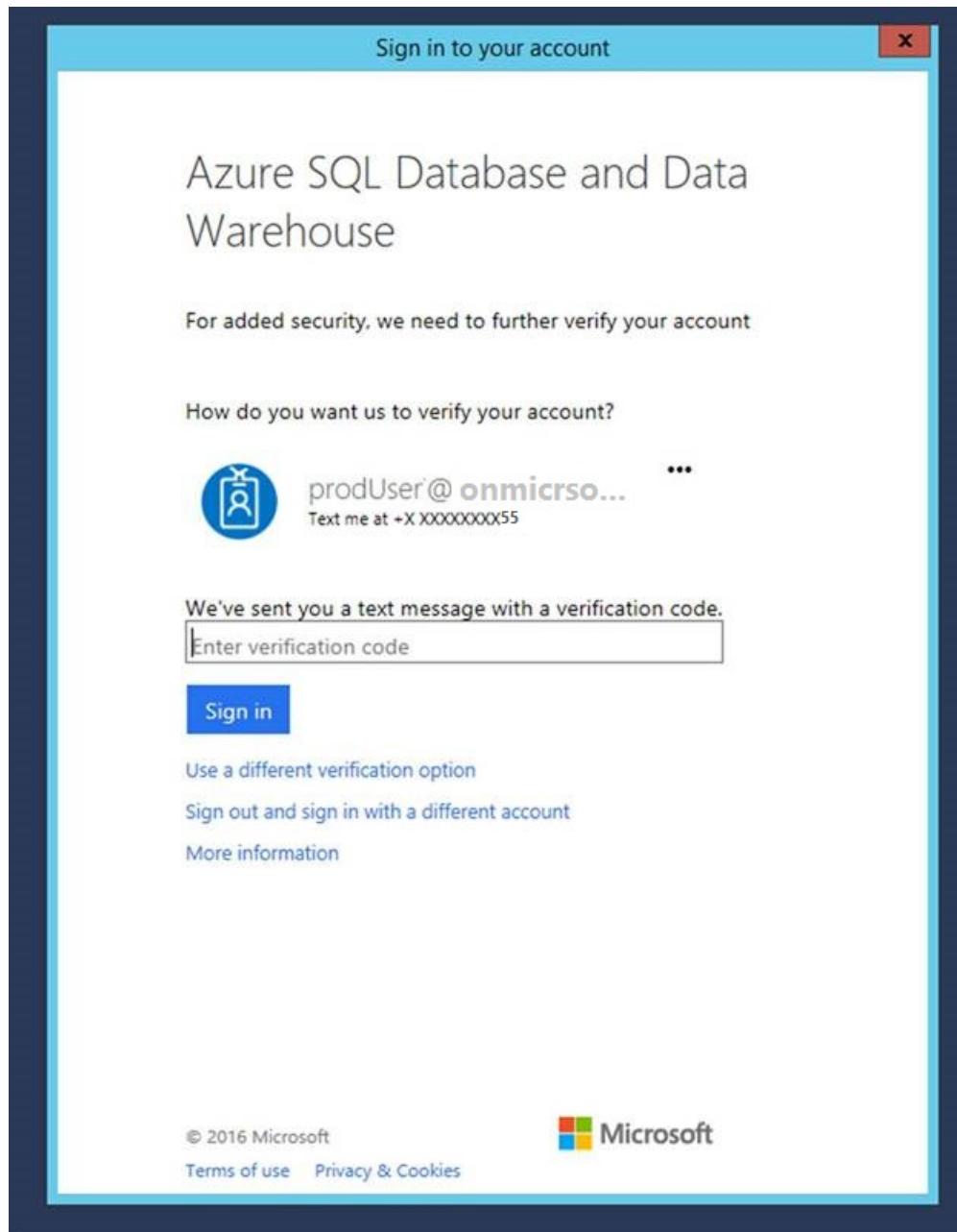
6. Two MFA setup dialog boxes might appear. This one time operation depends on the MFA administrator setting, and therefore may be optional. For an MFA enabled domain this step is sometimes pre-defined (for example, the domain requires users to use a smartcard and pin).



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 screenshot shows an 'Additional security verification' dialog box from Microsoft Azure. At the top, it says 'Microsoft Azure'. The main title is 'Additional security verification'. Below that, 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 for 'Authentication phone' set to 'United States (+1)' with the number '18005555555'. Below this is a 'Method' section with two radio buttons: 'Send me a code by text message' (selected) and 'Call me'. A large blue 'Contact me' button is at the bottom. A note at the bottom of the form area 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**.

# Get started with SQL database auditing

10/25/2018 • 10 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, OMS 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](#).

## 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.

You can configure auditing for different types of event categories, as explained in the [Set up auditing for your database](#) section.

## IMPORTANT

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.
- **Storage in VNet** is currently **not supported**.

## 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:

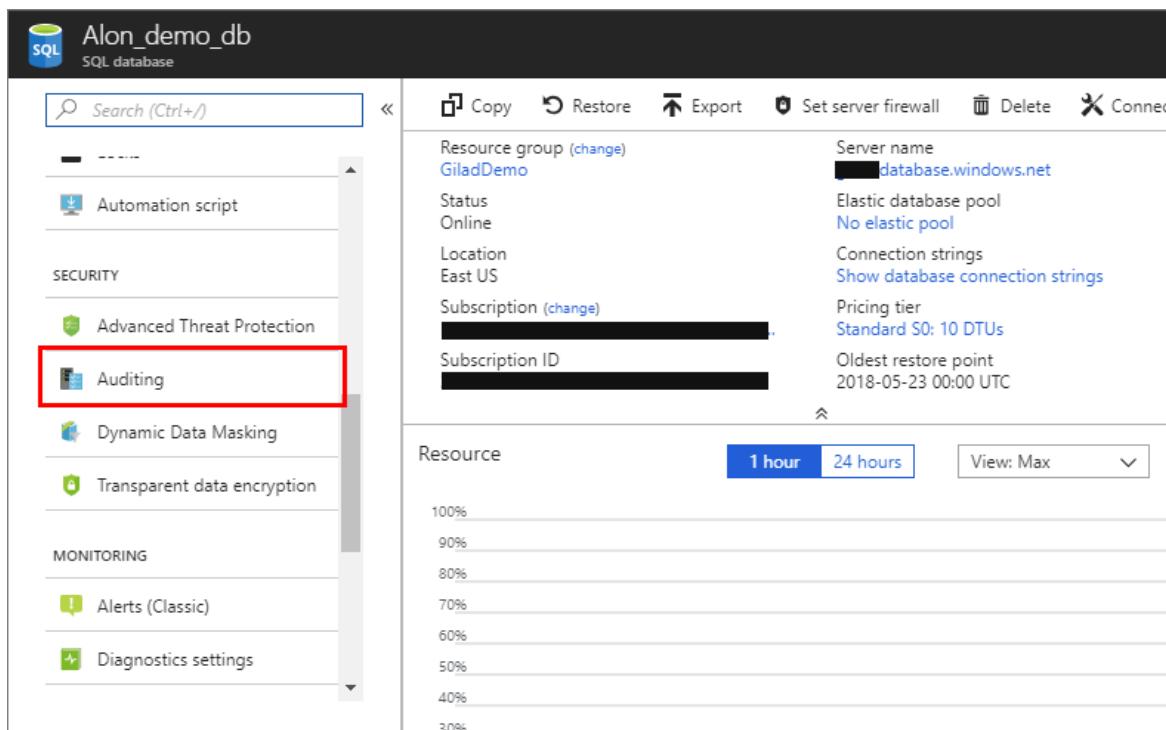
- You want to use a different *storage account* or *retention period* for a specific database.
- You want to audit event types or categories for a specific database that differ from the rest of the databases on the server. For example, you might have table inserts that need to be audited only for a specific database.

Otherwise, we recommended that you enable only server-level blob auditing and leave the database-level auditing disabled for all databases.

## Set up auditing for your database

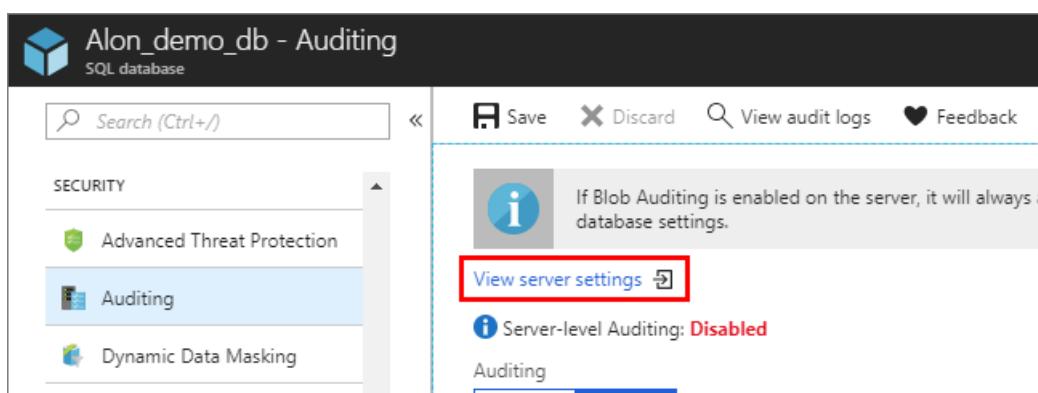
The following section describes the configuration of auditing using the Azure portal.

1. Go to the [Azure portal](#).
2. Navigate to **Auditing** under the Security heading in your SQL database/server pane.



The screenshot shows the Azure portal interface for a SQL database named 'Alon\_demo\_db'. The left sidebar has sections for SECURITY (Advanced Threat Protection, Auditing, Dynamic Data Masking, Transparent data encryption), MONITORING (Alerts (Classic), Diagnostics settings), and a general search bar. The main pane displays database details: Resource group (GildDemo), Server name (redacted database.windows.net), Status (Online), Location (East US), Subscription (redacted), Pricing tier (Standard S0: 10 DTUs), and Subscription ID (redacted). Below this is a resource usage chart showing usage over 1 hour and 24 hours, with a 'View: Max' dropdown. The 'Auditing' link in the sidebar is highlighted with a red box.

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.



The screenshot shows the 'Auditing' page for the 'Alon\_demo\_db' database. The left sidebar includes links for Advanced Threat Protection, Auditing (which is highlighted with a blue box), and Dynamic Data Masking. The main area contains a note: 'If Blob Auditing is enabled on the server, it will always affect database settings.' Below this is a 'View server settings' link (highlighted with a red box) and a status message: 'Server-level Auditing: Disabled'. There is also a 'View audit logs' and 'Feedback' button at the top right.

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.

Alon\_demo\_db - Auditing  
SQL database

Search (Ctrl+ /)

Automation script

SECURITY

Advanced Threat Protection

Auditing

Save Discard View audit logs Feedback

If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.

View server settings

Server-level Auditing: **Disabled**

Auditing

ON OFF

5. **New** - You now have multiple options for configuring where audit logs will be written. You can write logs to an Azure storage account, to a Log Analytics workspace for consumption by Log Analytics, 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.

Save Discard View audit logs Feedback

Learn more - Getting Started Guide

If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.

View server settings

Server-level Auditing: **Disabled**

Auditing

ON OFF

Audit log destination (choose at least one):

Storage

Storage details >

Log Analytics (Preview)

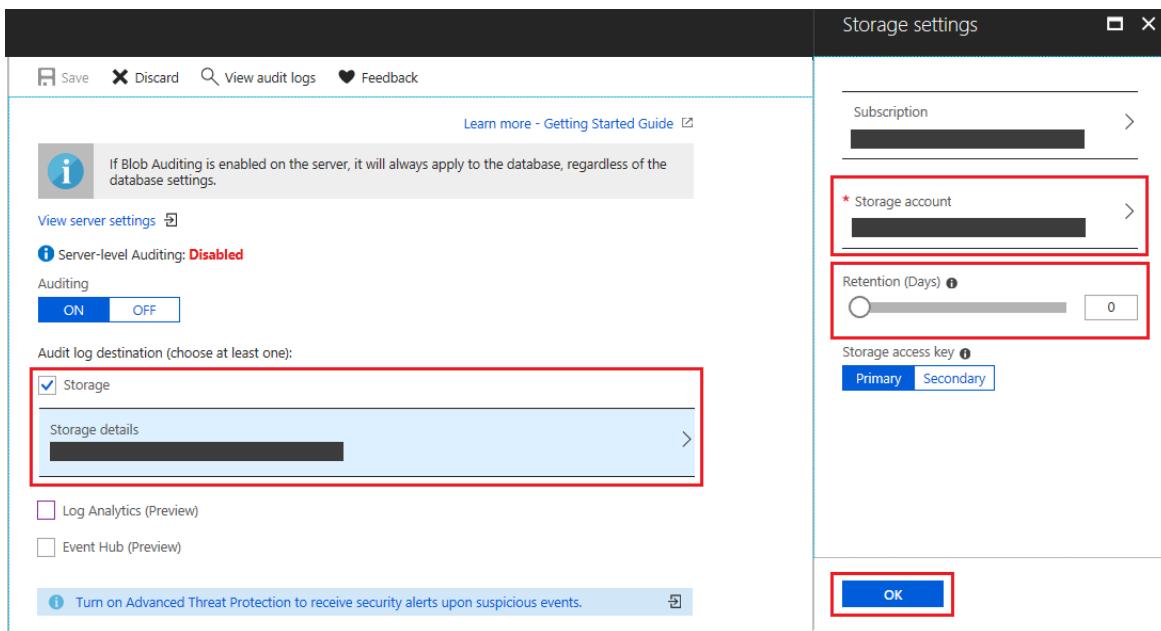
Log Analytics details >

Event Hub (Preview)

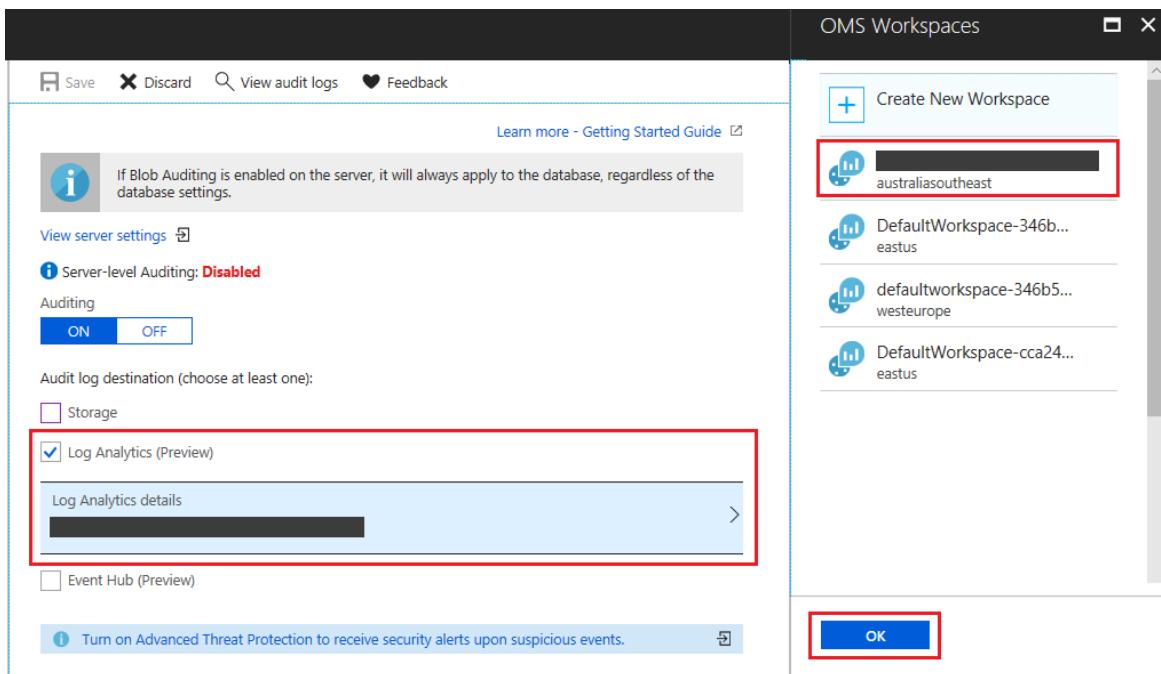
Event Hub details >

Turn on Advanced Threat Protection to receive security alerts upon suspicious events.

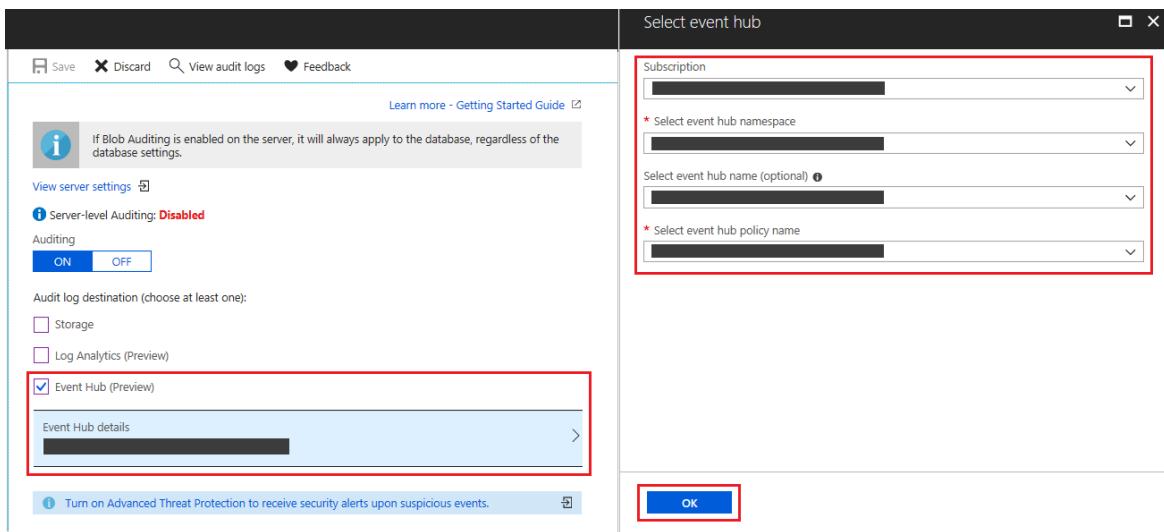
6. To configure writing audit logs to a storage account, select **Storage** and open **Storage details**. Select the Azure storage account where logs will be saved, and then select the retention period. The old logs will be deleted. Then click **OK**.



- 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**.



- 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.



9. Click **Save**.

10. If you want to customize the audited events, you can do this via [PowerShell cmdlets](#) or the [REST API](#).
11. 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](#).

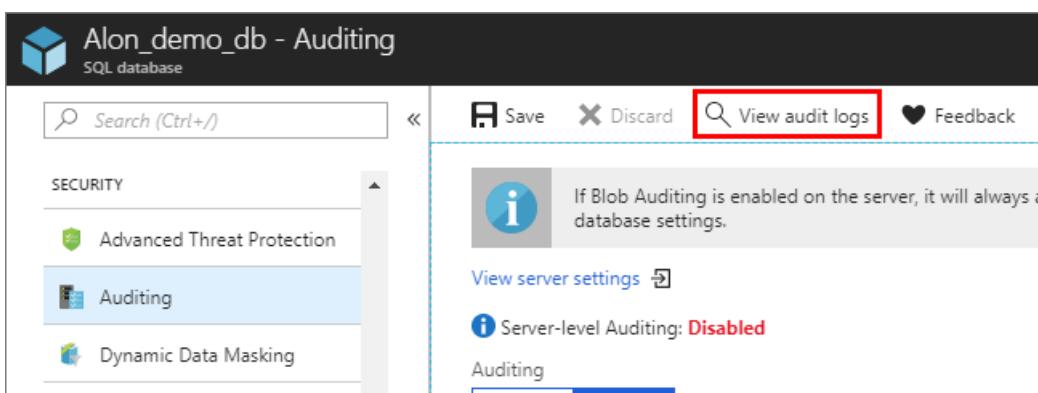
#### IMPORTANT

Enabling auditing on an Azure SQL Data Warehouse, or on a server that has an Azure SQL Data Warehouse on it, **will result in the Data Warehouse being resumed**, even in the case where it was previously paused. **Please make sure to pause the Data Warehouse again after enabling auditing.**

## Analyze audit logs and reports

If you chose to write audit logs to Log Analytics:

- Use the [Azure portal](#). Open the relevant database. At the top of the database's **Auditing** page, click **View audit logs**.



- Then, clicking on **Open in OMS** at the top of the **Audit records** page will open the Logs view in Log Analytics, where you can customize the time range and the search query.

The screenshot shows the 'Audit records' page for the database 'Alon\_demo\_db'. At the top, there are buttons for 'Refresh', 'Filter', and 'Open in OMS' (which is highlighted with a red box). Below this is an information icon with a link to learn more about viewing and analyzing audit records. A section for 'Audit source' includes tabs for 'Server audit' (selected) and 'Database audit'. There is also 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 [Log Analytics](#) to run advanced searches on your audit log data. Log Analytics 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 Log Analytics search language and commands, see [Log Analytics 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](#).

If you chose to write audit logs to an Azure storage account, there are several methods you can use to view the logs:

- 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 **sqlauditlogs**. For further details about the hierarchy of the storage folder, naming conventions, and log format, see the [Blob Audit Log Format Reference](#).
- 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 'Auditing' page for the database 'Alon\_demo\_db'. On the left, there is a sidebar with options: 'Search (Ctrl+)', 'Save', 'Discard', 'View audit logs' (which is highlighted with a red box), and 'Feedback'. The main area displays information about Blob Auditing being enabled on the server, with a note that it will always affect database settings. It also shows that Server-level Auditing is disabled. A progress bar indicates that Auditing is in progress.

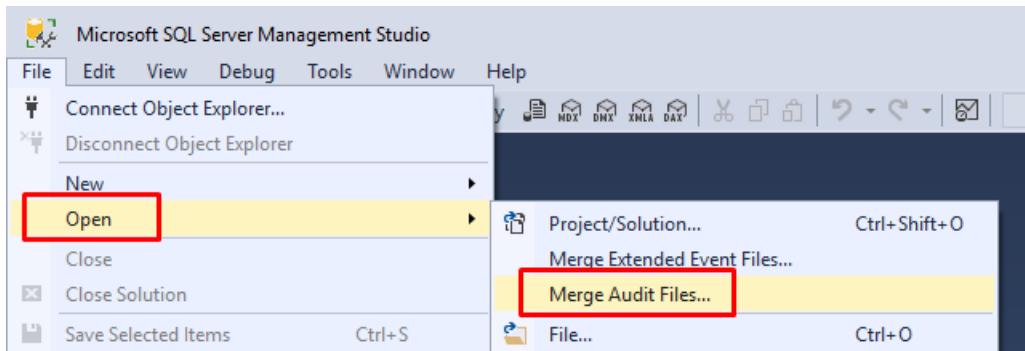
**Audit records** opens, from which you'll be able to view the logs.

- 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.

- 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**.



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:
      - Use the [Extended Events Reader C# library](#).
      - [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.

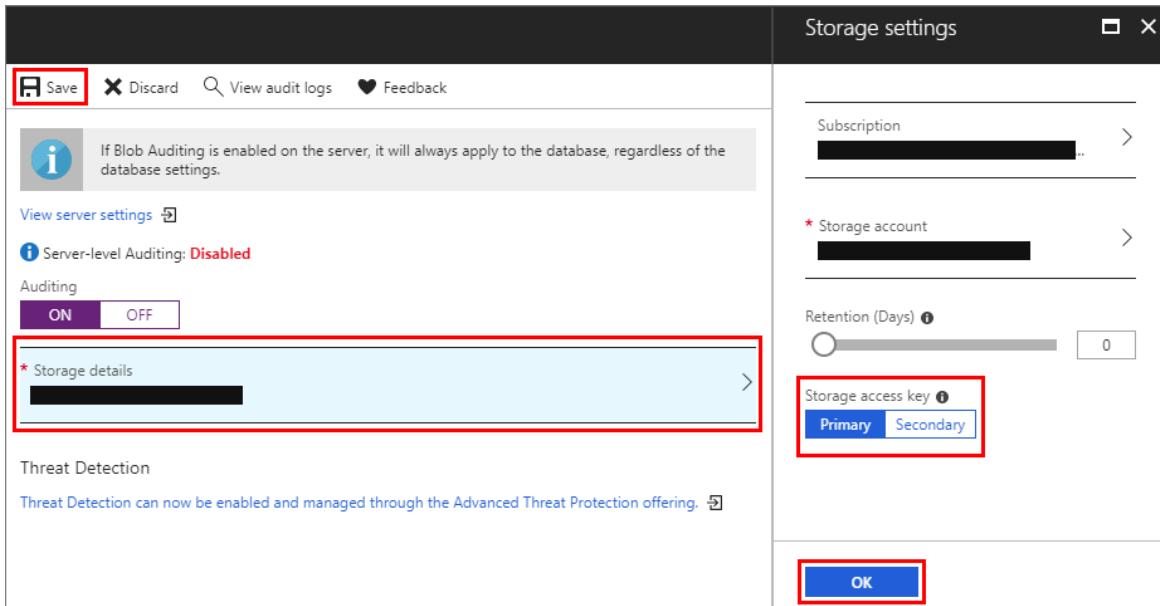
- Server-level (**recommended**): Turn on auditing on both the **primary server** as well as the **secondary server** - the primary and secondary databases will each be audited independently based on their respective server-level policy.
- Database-level: Database-level auditing for secondary databases 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

- 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.
  - **Storage in VNet** is currently **not supported**.
- 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.

# Manage SQL database auditing using Azure PowerShell

## **PowerShell cmdlets:**

- [Create or Update Database Blob Auditing Policy \(Set-AzureRMSqlDatabaseAuditing\)](#)
- [Create or Update Server Blob Auditing Policy \(Set-AzureRMSqlServerAuditing\)](#)
- [Get Database Auditing Policy \(Get-AzureRMSqlDatabaseAuditing\)](#)
- [Get Server Blob Auditing Policy \(Get-AzureRMSqlServerAuditing\)](#)

For a script example, see [Configure auditing and threat detection using PowerShell](#).

# Manage SQL database auditing using REST API

## **REST API - Blob auditing:**

- [Create or Update Database Blob Auditing Policy](#)
- [Create or Update Server Blob Auditing Policy](#)
- [Get Database Blob Auditing Policy](#)
- [Get Server Blob Auditing Policy](#)

Extended policy with WHERE clause support for additional filtering:

- [Create or Update Database \*Extended\* Blob Auditing Policy](#)
- [Create or Update Server \*Extended\* Blob Auditing Policy](#)
- [Get Database \*Extended\* Blob Auditing Policy](#)
- [Get Server \*Extended\* Blob Auditing Policy](#)

# Get started with Azure SQL Database Managed Instance Auditing

10/11/2018 • 3 minutes to read • [Edit Online](#)

Azure SQL Database 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](#).

## Set up auditing for your server

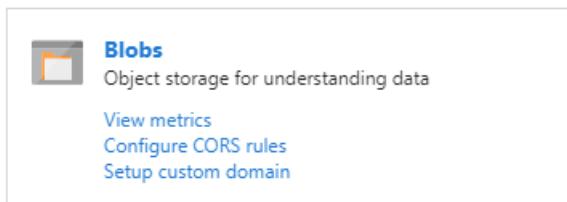
The following section describes the configuration of auditing on your Managed Instance.

1. Go to the [Azure portal](#).
2. The following steps create an Azure Storage **container** where audit logs are stored.
  - 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 server to avoid cross-region reads/writes.

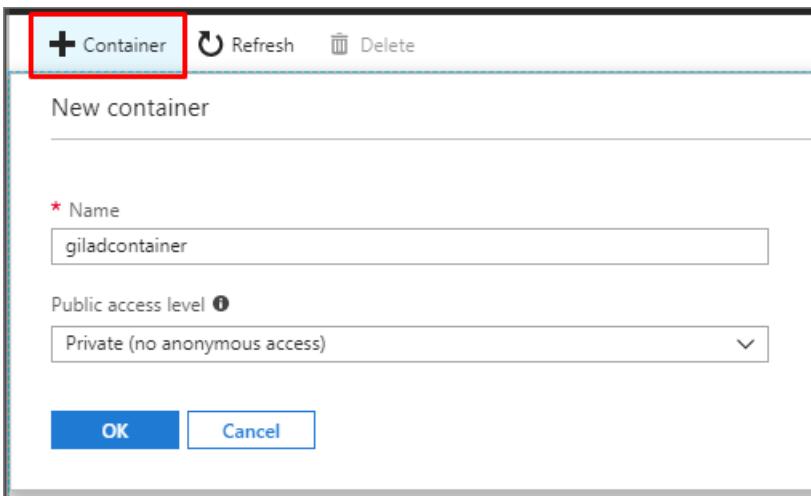
- In the storage account, go to **Overview** and click **Blobs**.



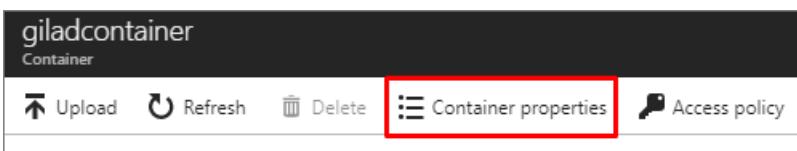
- In the top menu, click **+ Container** to create a new container.



- Provide a container **Name**, set Public access level to **Private**, and then click **OK**.

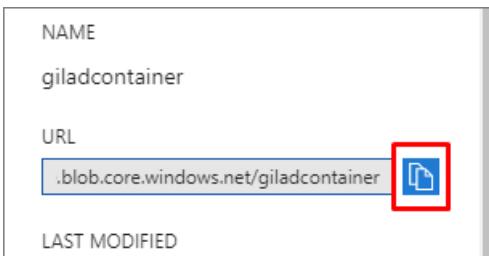


- In the containers list, click the newly created container and then click **Container properties**.



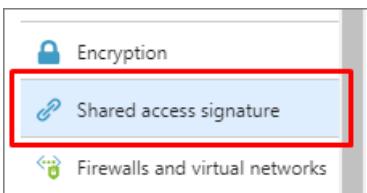
- Copy the container URL by clicking the copy icon and save the URL (for example, in Notepad) for future use. The container URL format should be

`https://<StorageName>.blob.core.windows.net/<ContainerName>`



- The following steps generate an Azure Storage **SAS Token** used to grant Managed Instance Auditing access rights to the storage account.

- Navigate to the Azure Storage account where you created the container in the previous step.
- Click on **Shared access signature** in the Storage Settings menu.



- Configure the SAS as follows:
  - Allowed services:** Blob
  - Start date:** to avoid time zone-related issues, it is recommended to use yesterday's date.
  - End date:** choose the date on which this SAS Token expires.

**NOTE**

Renew the token upon expiry to avoid audit failures.

- Click **Generate SAS**.

**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

End  
    
(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** ⓘ

**Generate SAS**

- 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.

**Generate SAS**

**SAS token** ⓘ

?sv=2017-07-29&ss=b&srt=sco&sp=rwdrllac&se=2018-03-19T18:03:39Z&st=2018-03-...

4. Connect to your Managed Instance via SQL Server Management Studio (SSMS).

5. 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
```

6. 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):

```
CREATE SERVER AUDIT [<your_audit_name>]
TO URL ( PATH = '<container_url>' [, RETENTION_DAYS = integer ])
GO
```

If not specified, `RETENTION_DAYS` default is 0 (unlimited retention).

For additional information:

- [Auditing differences between Managed Instance, Azure SQL DB and SQL Server](#)
- [CREATE SERVER AUDIT](#)
- [ALTER SERVER AUDIT](#)

7. 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](#)

8. Enable the server audit that you created in step 6:

```
ALTER SERVER AUDIT [<your_audit_name>]
WITH (STATE=ON);
GO
```

## Analyze audit logs

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](#).
- For a full list of audit log consumption methods, refer to the [Get started with SQL database auditing] (<https://docs.microsoft.com/azure/sql-database/sql-database-auditing>).

### IMPORTANT

The method for viewing audit records from the Azure portal ('Audit records' pane) is currently unavailable for Managed Instance.

## Auditing differences between Managed Instance, Azure SQL Database, and SQL Server

The key differences between SQL Audit in Managed Instance, Azure SQL Database, and SQL Server on-premises are:

- In Managed Instance, SQL Audit works at the server level and stores `.xe1` log files on Azure blob storage account.
- In Azure SQL Database, SQL Audit works at the database level.
- In SQL Server on-premises / virtual machines, SQL 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 `.xe1` files are placed.
- The syntax `TO FILE` is **not supported** because Managed Instance 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] (<https://docs.microsoft.com/azure/sql-database/sql-database-auditing>).
- For more information about Azure programs that support standards compliance, see the [Azure Trust Center](#).

# SQL Database - Downlevel clients support and IP endpoint changes for Table Auditing

10/5/2018 • 2 minutes to read • [Edit Online](#)

## IMPORTANT

This document applies only to Table Auditing, which is **now deprecated**.

Please use the new [Blob Auditing](#) method, which **does not** require downlevel client connection string modifications.

Additional info on Blob Auditing can be found in [Get started with SQL database auditing](#).

[Database Auditing](#) works automatically with SQL clients that support TDS redirection. Note that redirection does not apply when using the Blob Auditing method.

## Downlevel clients support

Any client which implements TDS 7.4 should also support redirection. Exceptions to this include JDBC 4.0 in which the redirection feature is not fully supported and Tedious for Node.JS in which redirection was not implemented.

For "Downlevel clients", i.e. which support TDS version 7.3 and below - the server FQDN in the connection string should be modified:

Original server FQDN in the connection string: <*server name*>.database.windows.net

Modified server FQDN in the connection string: <*server name*>.database.**secure**.windows.net

A partial list of "Downlevel clients" includes:

- .NET 4.0 and below,
- ODBC 10.0 and below.
- JDBC (while JDBC does support TDS 7.4, the TDS redirection feature is not fully supported)
- Tedious (for Node.JS)

**Remark:** The above server FDQN modification may be useful also for applying a SQL Server Level Auditing policy without a need for a configuration step in each database (Temporary mitigation).

## IP endpoint changes when enabling Auditing

Please note that when you enable Table Auditing, the IP endpoint of your database will change. If you have strict firewall settings, please update those firewall settings accordingly.

The new database IP endpoint will depend on the database region:

| DATABASE REGION | POSSIBLE IP ENDPOINTS                                       |
|-----------------|---|
| China North     | 139.217.29.176, 139.217.28.254                              |
| China East      | 42.159.245.65, 42.159.246.245                               |
| Australia East  | 104.210.91.32, 40.126.244.159, 191.239.64.60, 40.126.255.94 |

| DATABASE REGION     | POSSIBLE IP ENDPOINTS  |
|---------------------|--|
| Australia Southeast | 191.239.184.223, 40.127.85.81, 191.239.161.83, 40.127.81.130   |
| Brazil South        | 104.41.44.161, 104.41.62.230, 23.97.99.54, 104.41.59.191   |
| Central US          | 104.43.255.70, 40.83.14.7, 23.99.128.244, 40.83.15.176   |
| Central US EUAP     | 52.180.178.16, 52.180.176.190  |
| East Asia           | 23.99.125.133, 13.75.40.42, 23.97.71.138, 13.94.43.245   |
| East US 2           | 104.209.141.31, 104.208.238.177, 191.237.131.51, 104.208.235.50  |
| East US             | 23.96.107.223, 104.41.150.122, 23.96.38.170, 104.41.146.44   |
| East US EUAP        | 52.225.190.86, 52.225.191.187  |
| Central India       | 104.211.98.219, 104.211.103.71   |
| South India         | 104.211.227.102, 104.211.225.157   |
| West India          | 104.211.161.152, 104.211.162.21  |
| Japan East          | 104.41.179.1, 40.115.253.81, 23.102.64.207, 40.115.250.196   |
| Japan West          | 104.214.140.140, 104.214.146.31, 191.233.32.34, 104.214.146.198  |
| North Central US    | 191.236.155.178, 23.96.192.130, 23.96.177.169, 23.96.193.231   |
| North Europe        | 104.41.209.221, 40.85.139.245, 137.116.251.66, 40.85.142.176   |
| South Central US    | 191.238.184.128, 40.84.190.84, 23.102.160.153, 40.84.186.66  |
| Southeast Asia      | 104.215.198.156, 13.76.252.200, 23.97.51.109, 13.76.252.113  |
| West Europe         | 104.40.230.120, 13.80.23.64, 137.117.171.161, 13.80.8.37, 104.47.167.215, 40.118.56.193, 104.40.176.73, 40.118.56.20 |
| West US             | 191.236.123.146, 138.91.163.240, 168.62.194.148, 23.99.6.91  |
| West US 2           | 13.66.224.156, 13.66.227.8   |
| West Central US     | 52.161.29.186, 52.161.27.213   |
| Canada Central      | 13.88.248.106, 13.88.248.110   |

| DATABASE REGION | POSSIBLE IP ENDPOINTS       |
|-----------------|-----------------------------|
| Canada East     | 40.86.227.82, 40.86.225.194 |
| UK North        | 13.87.101.18, 13.87.100.232 |
| UK South 2      | 13.87.32.202, 13.87.32.226  |

# SQL Database dynamic data masking

9/25/2018 • 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.

## SQL Database 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.

### Dynamic data masking permissions

Dynamic data masking can be configured by the Azure Database admin, server admin, or security officer 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  |
|------------------|--|
| <b>Default</b>   | <p><b>Full masking according to the data types of the designated fields</b></p> <ul style="list-style-type: none"><li>• Use XXXX or fewer Xs if the size of the field is less than 4 characters for string data types (nchar, ntext, nvarchar).</li><li>• Use a zero value for numeric data types (bigint, bit, decimal, int, money, numeric, smallint, smallmoney, tinyint, float, real).</li><li>• Use 01-01-1900 for date/time data types (date, datetime2, datetime, datetimeoffset, smalldatetime, time).</li><li>• For SQL variant, the default value of the current type is used.</li><li>• For XML the document is used.</li><li>• Use an empty value for special data types (timestamp table, hierarchyid, GUID, binary, image, varbinary spatial types).</li></ul> |

| MASKING FUNCTION                                 | MASKING LOGIC   |  |  |  |  |  |  |
|--|---|--|--|--|--|--|--|
| Credit card                                      | <p><b>Masking method, which exposes the last four digits of the designated fields</b> and adds a constant string as a prefix in the form of a credit card.</p> <p>XXXX-XXXX-XXXX-1234</p>   |  |  |  |  |  |  |
| Email  | <p><b>Masking method, which exposes the first letter and replaces the domain with XXX.com</b> using a constant string prefix in the form of an email address.</p> <p>aXX@XXXX.com</p>   |  |  |  |  |  |  |
| Random number                                    | <p><b>Masking method, which generates a random number</b> 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<br/> <input type="button" value="Random number"/>▼</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">From<br/><input type="text" value="0"/></td> <td style="width: 50%;">To<br/><input type="text" value="0"/></td> </tr> <tr> <td style="text-align: center;"></td> <td style="text-align: center;"></td> </tr> </table>   | From<br><input type="text" value="0"/>           | To<br><input type="text" value="0"/>                 |  |  |  |  |
| From<br><input type="text" value="0"/>           | To<br><input type="text" value="0"/>  |  |  |  |  |  |  |
|  |   |  |  |  |  |  |  |
| Custom text                                      | <p><b>Masking method, which exposes the first and last characters</b> 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.<br/>prefix[padding]suffix</p> <p>Masking Field Format<br/> <input type="button" value="Custom text"/>▼</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%;">Exposed Prefix<br/><input type="text" value="3"/></td> <td style="width: 33%;">Padding String<br/><input type="text" value="X*X*X"/></td> <td style="width: 33%;">Exposed Suffix<br/><input type="text" value="2"/></td> </tr> <tr> <td style="text-align: center;"></td> <td style="text-align: center;"></td> <td style="text-align: center;"></td> </tr> </table> | Exposed Prefix<br><input type="text" value="3"/> | Padding String<br><input type="text" value="X*X*X"/> | Exposed Suffix<br><input type="text" value="2"/> |  |  |  |
| Exposed Prefix<br><input type="text" value="3"/> | Padding String<br><input type="text" value="X*X*X"/>  | Exposed Suffix<br><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](#).

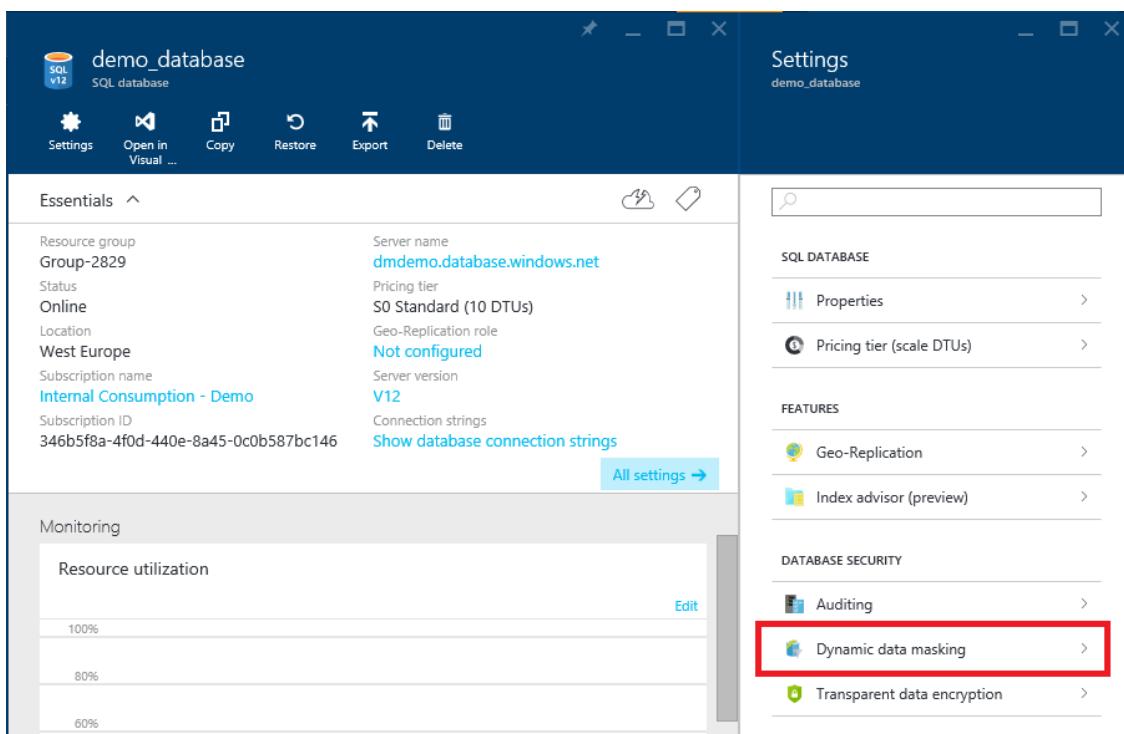
# Get started with SQL Database dynamic data masking with the Azure portal

9/25/2018 • 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 note says 'Downlevel clients require the use of Security Enabled Connection Strings.' Below this, the 'Masking Rules' section is empty, showing the message 'You haven't created any masking rules.' A note below it states 'SQL users excluded from masking (administrators are always excluded)' with a checkmark. The 'Recommended fields to mask' section is highlighted with a red border and lists the following:

| 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 'Dynamic Data Masking' interface for the 'demo\_database'. At the top, there are three buttons: 'Save', 'Discard', and 'Add Mask', with 'Add Mask' highlighted by a red box. A message bar at the top states: 'Downlevel clients require the use of Security Enabled Connection Strings.' Below this is a section titled 'Masking Rules' with two columns: 'MASK NAME' and 'MASK FUNCTION'. Three rules are listed:

| MASK NAME                     | MASK FUNCTION                       |
|-------------------------------|-------------------------------------|
| SalesLT_Customer_LastName     | Default value (0, xxxx, 01-01-1900) |
| SalesLT_Customer_EmailAddress | Default value (0, xxxx, 01-01-1900) |
| SalesLT_Customer_Phone        | Default value (0, xxxx, 01-01-1900) |

Below the rules, a note says: 'SQL users excluded from masking (administrators are always excluded)'. A tooltip for this note is shown: 'SQL users excluded from masking (administrators are always excluded)' with a green checkmark icon.

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.

The screenshot shows the 'Add Masking Rule' dialog. At the top, there are three buttons: 'Save', 'Discard', and 'Delete'. The 'Mask Name' field contains 'SalesLT\_SalesOrderHeader\_AccountNumber'. The 'Select what to mask' section includes dropdowns for 'Schema' (set to 'SalesLT'), 'Table' (set to 'SalesOrderHeader'), and 'Column' (set to 'AccountNumber (nvarchar)'). Below this, the 'Select how to mask' section contains a list of masking formats. One item, 'Credit card value (xxxx-xxxx-xxxx-1234)', is highlighted with a blue background and a red box around the entire list.

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](#).

# Always Encrypted: Protect sensitive data and store encryption keys in the Windows certificate store

10/8/2018 • 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.

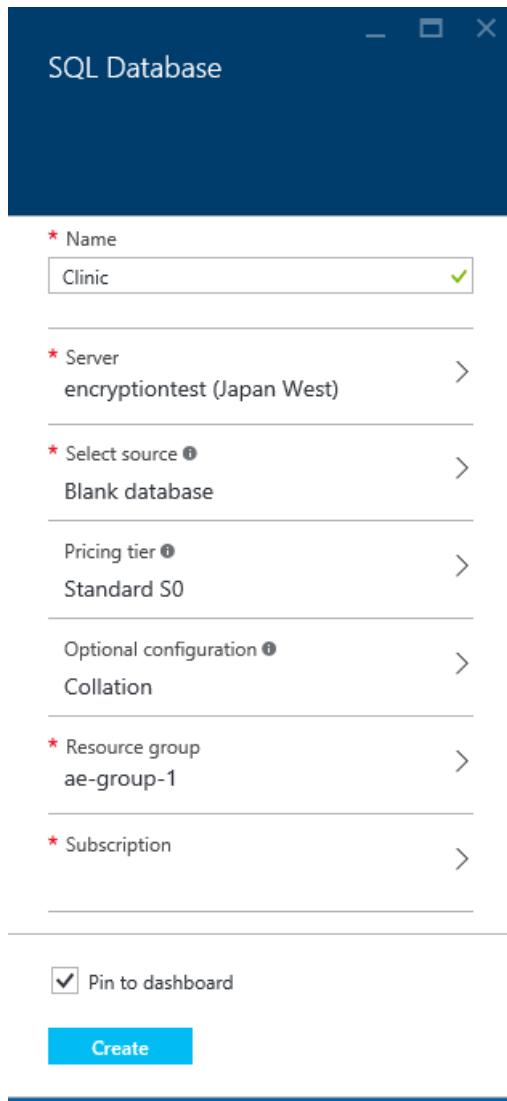
## Prerequisites

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**.

ADO.NET

```
Server=tcp:encryptiontest.database.windows.net,1433;Database=Clinic;User ID=sstein@encn...
```

ODBC (Includes Node.js)

```
Driver=(SQL Server Native Client 11.0);Server=tcp:encryptiontest.database.windows.net,1433;
```

PHP

```
Server: encryptiontest.database.windows.net,1433 \nSQL Database: Clinic\nUser Name: ss
```

JDBC

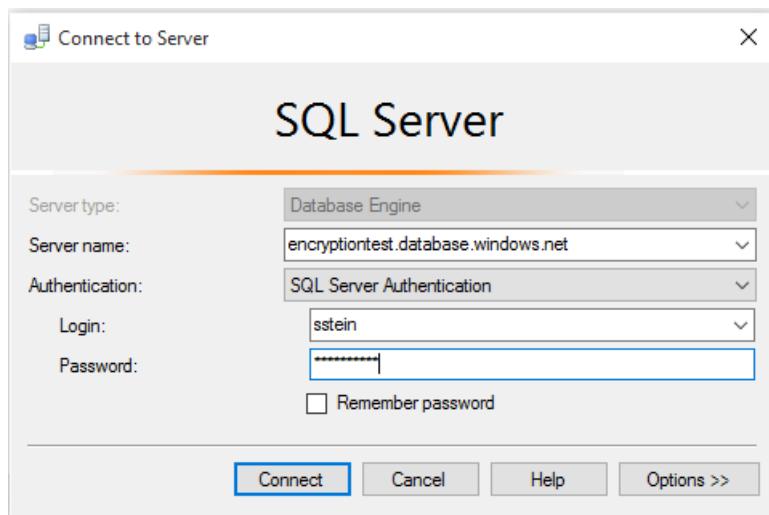
```
jdbc:sqlserver://encryptiontest.database.windows.net:1433;database=Clinic;user=sstein@enc...
```

## 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.

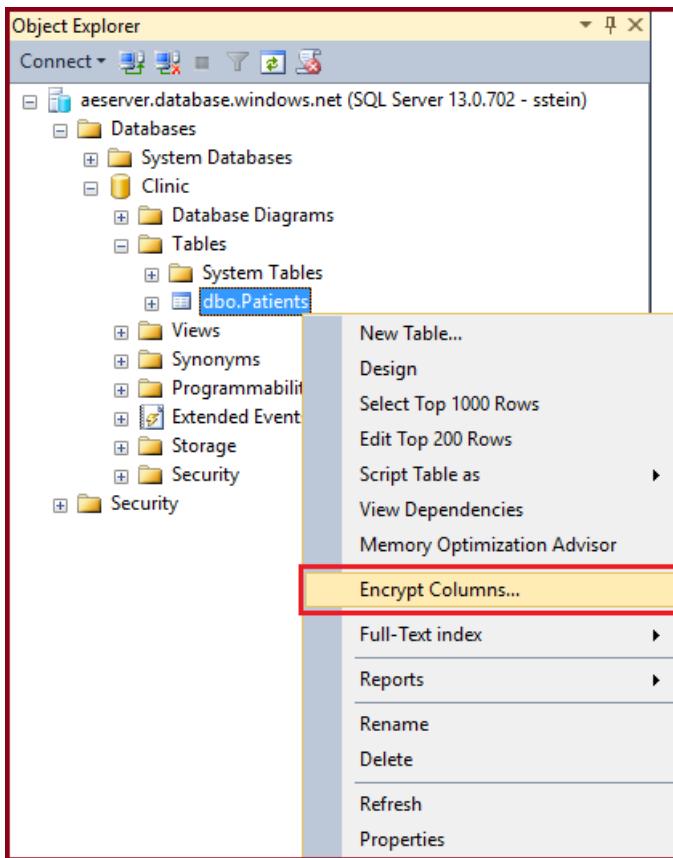
1. Expand **Databases**.
2. Right-click the **Clinic** database and click **New Query**.
3. Paste the following Transact-SQL (T-SQL) into the new query window and **Execute** it.

```
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:



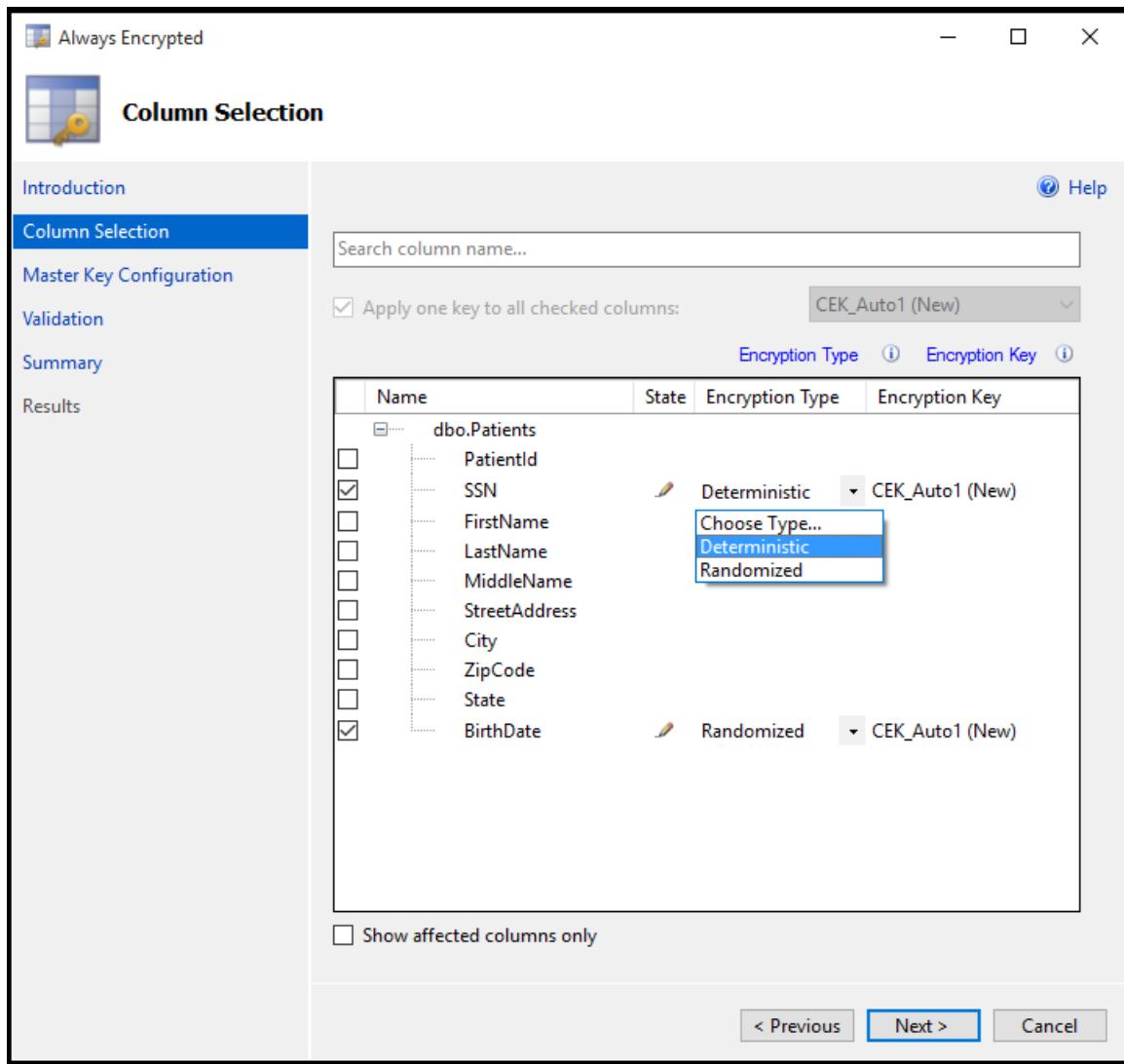
The Always Encrypted wizard includes the following sections: **Column Selection**, **Master Key Configuration (CMK)**, **Validation**, and **Summary**.

#### 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.

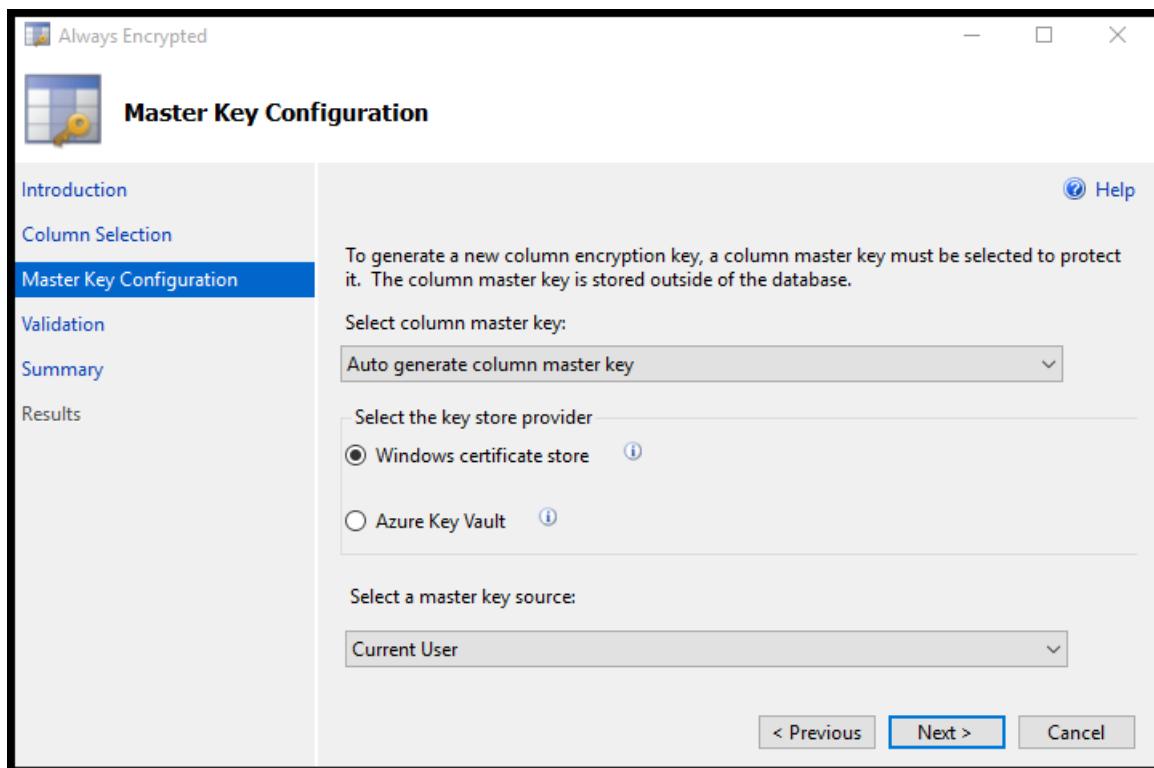
Set the **Encryption Type** for the **SSN** column to **Deterministic** and the **BirthDate** column to **Randomized**. Click **Next**.



## 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**.

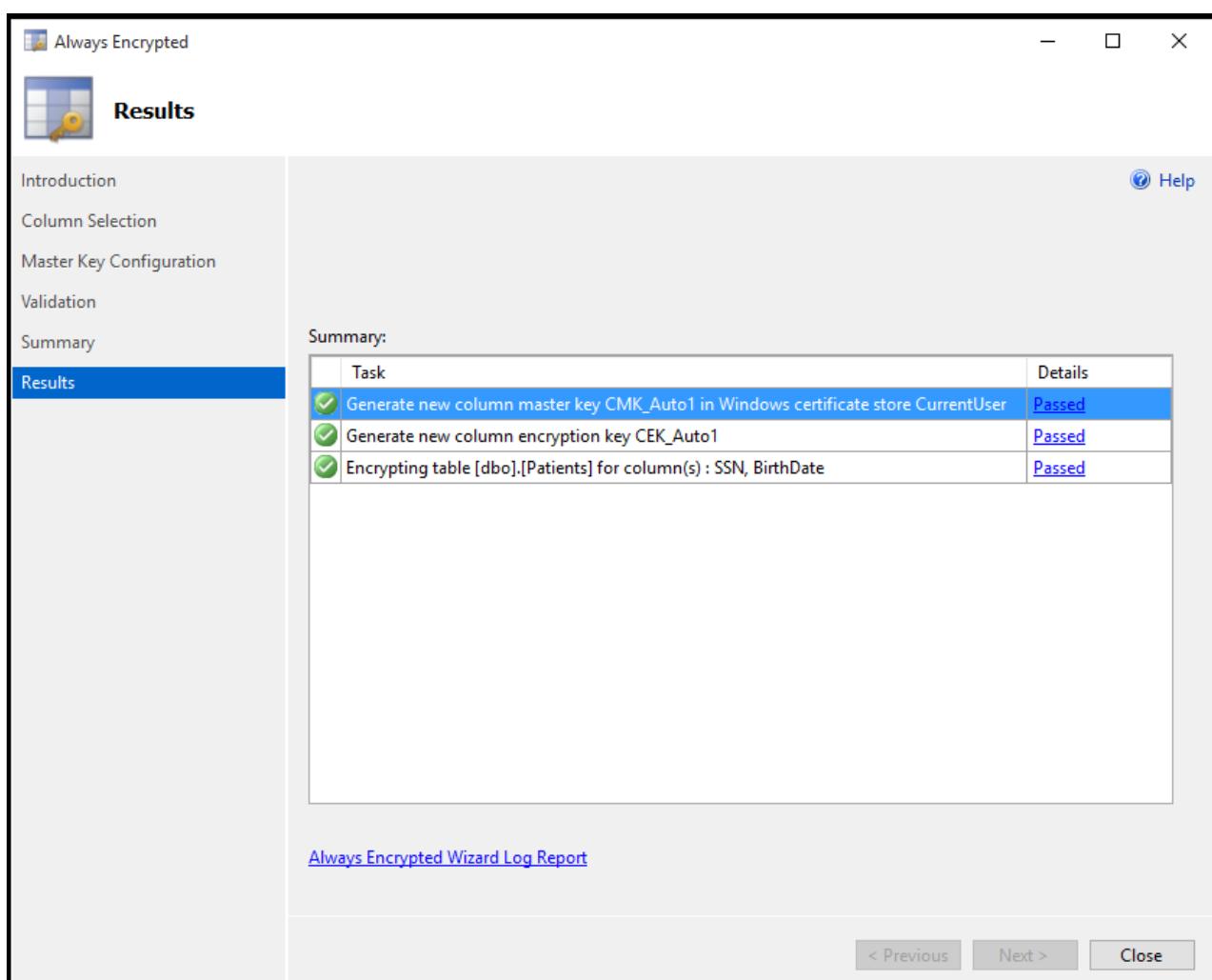


## Validation

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.

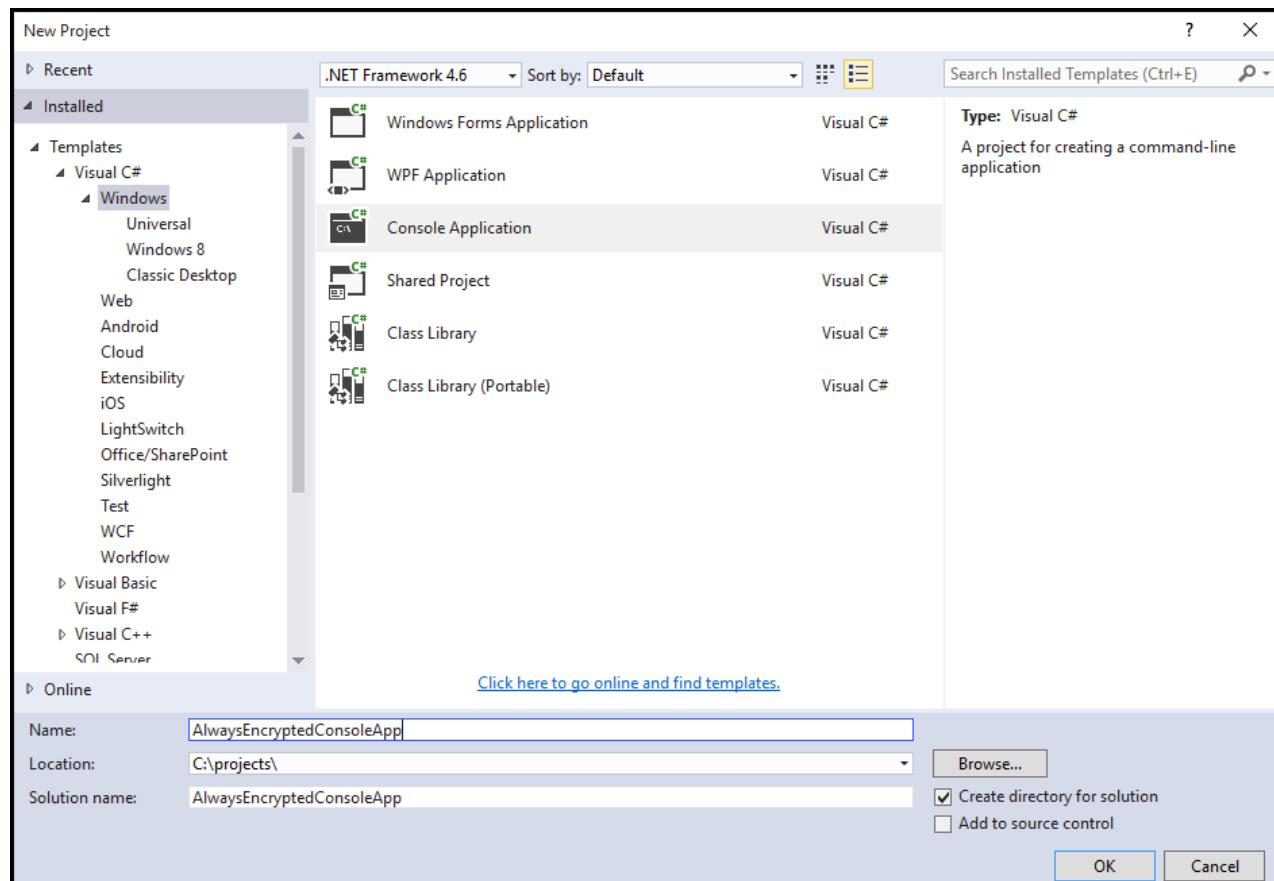
## Create a client application that works with the encrypted data

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.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Data;  
using System.Data.SqlClient;  
  
namespace AlwaysEncryptedConsoleApp  
{
```

```

class Program
{
    // Update this line with your Clinic database connection string from the Azure portal.
    static string connectionString = @"Replace with your connection string";

    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...");

        InsertPatient(new Patient() {
            SSN = "999-99-0001", FirstName = "Orlando", 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 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();
                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; }
}
}

```

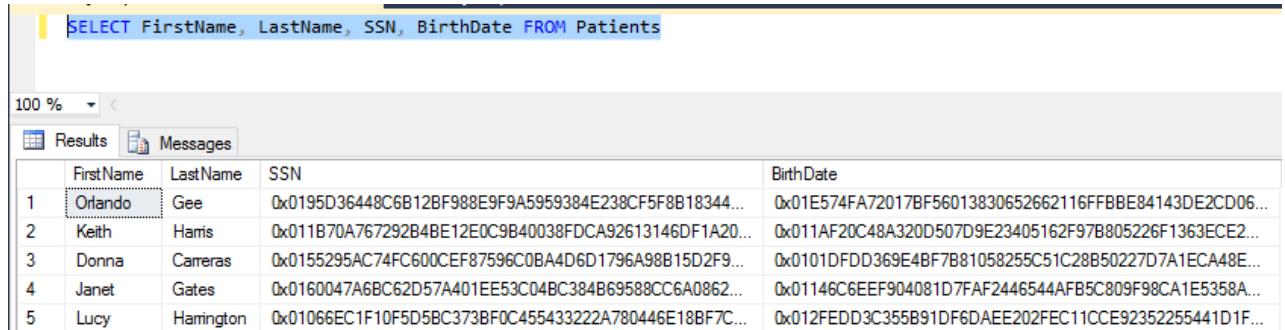
## Verify that the data is encrypted

You can quickly check that the actual data on the server is encrypted by querying the **Patients** data with SSMS. (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.

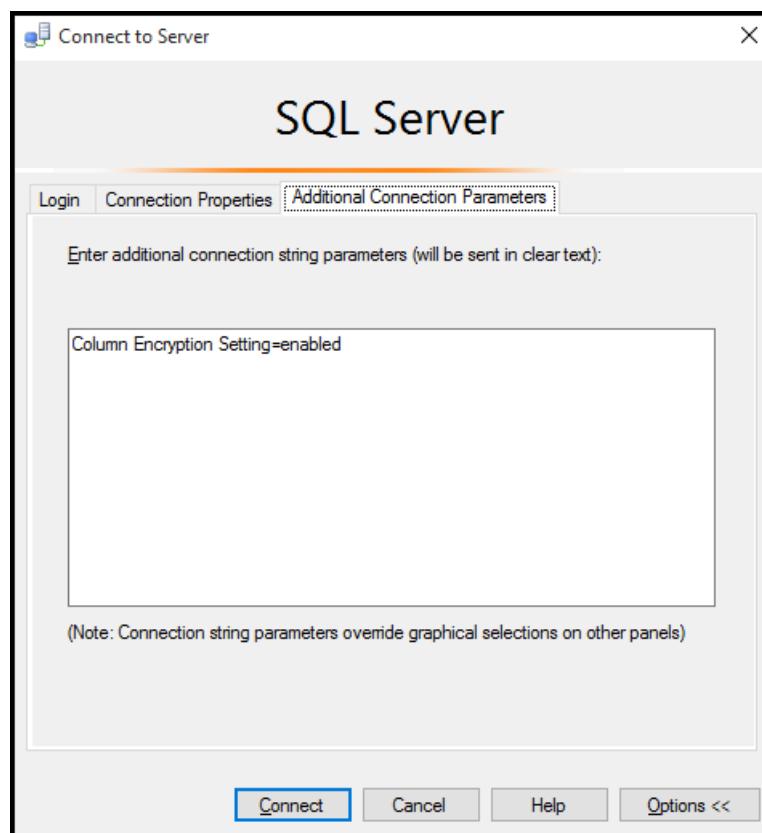


The screenshot shows the SSMS interface with a query window containing the SQL command: `SELECT FirstName, LastName, SSN, BirthDate FROM Patients`. Below the query window is a results grid. The grid has columns: FirstName, LastName, SSN, and BirthDate. There are 5 rows of data:

|   | FirstName | LastName  | SSN  | BirthDate  |
|---|-----------|-----------|--|--|
| 1 | Orlando   | Gee       | 0x0195D36448C6B12BF988E9F9A5959384E238CF5F8B18344... | 0x01E574FA72017BF56013830652662116FFBBE84143DE2CD06... |
| 2 | Keith     | Hamis     | 0x011B70A767292B4BE12E0C9B40038FDCA92613146DF1A20... | 0x011AF20C48A320D507D9E23405162F97B805226F1363ECE2...  |
| 3 | Donna     | Cameras   | 0x0155295AC74FC600CEF87596C08A4D6D1796A98B15D2F9...  | 0x0101DFDD369E4BF7B81058255C51C28B50227D7A1ECA48E...   |
| 4 | Janet     | Gates     | 0x0160047A6BC62D57A401EE53C04BC384B69588CC6A0862...  | 0x01146C6EEF904081D7FAF2446544AFB5C809F98CA1E5358A...  |
| 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**.



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     | Harris    | 999-99-0002 | 1977-06-20 |
| 3 | Donna     | Cameras   | 999-99-0003 | 1973-02-09 |
| 4 | Janet     | Gates     | 999-99-0004 | 1985-08-31 |
| 5 | Lucy      | Hamington | 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

10/23/2018 • 13 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](#), version 1.0 or later. Type **(Get-Module azure -ListAvailable).Version** to see what version of PowerShell you are running.

## 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).

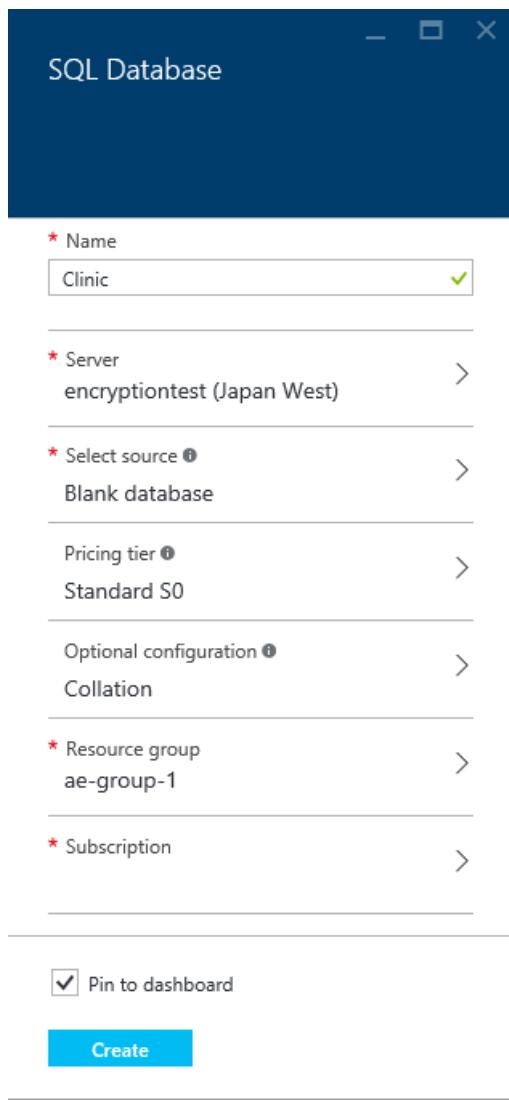
The *create*, *get*, *list*, *sign*, *verify*, *wrapKey*, and *unwrapKey* permissions are required for creating a new column master key and for setting up encryption with SQL Server Management Studio.

You can quickly create a key vault by running the following script. For a detailed explanation of these cmdlets and more information about creating and configuring a key vault, see [Get started with Azure Key Vault](#).

```
$subscriptionName = '<your Azure subscription name>'  
$userPrincipalName = '<username@domain.com>'  
$applicationId = '<application ID from your AAD application>'  
$resourceGroupName = '<resource group name>'  
$location = '<datacenter location>'  
$vaultName = 'AeKeyVault'  
  
Connect-AzureRmAccount  
$subscriptionId = (Get-AzureRmSubscription -SubscriptionName $subscriptionName).Id  
Set-AzureRmContext -SubscriptionId $subscriptionId  
  
New-AzureRmResourceGroup -Name $resourceGroupName -Location $location  
New-AzureRmKeyVault -VaultName $vaultName -ResourceGroupName $resourceGroupName -Location $location  
  
Set-AzureRmKeyVaultAccessPolicy -VaultName $vaultName -ResourceGroupName $resourceGroupName -  
PermissionsToKeys create,get,wrapKey,unwrapKey,sign,verify,list -UserPrincipalName $userPrincipalName  
Set-AzureRmKeyVaultAccessPolicy -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](#).



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**.

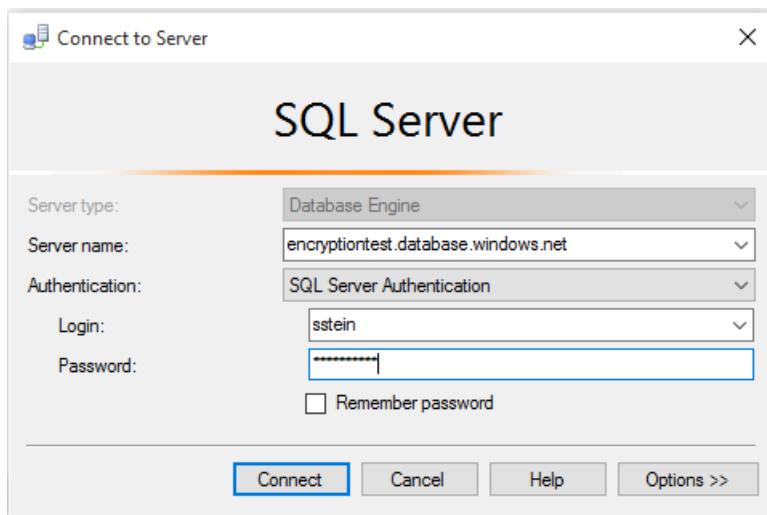
The screenshot shows the 'Clinic' database details page. It includes settings like 'Settings', 'Open in Visual ...', 'Copy', 'Restore', and 'Delete'. The 'Essentials' section shows the resource group 'ae-group-1', status 'Online', location 'Japan West', and subscription 'Encryptiontest'. It also lists the 'Subscription ID' and 'Monitoring' status. Below this, the 'Connection strings' section is highlighted, showing the ADO.NET connection string: 'Server=tcp:encryptiontest.database.windows.net,1433;Database=Clinic;User ID=sstein@encn...'. Other sections shown include ODBC, PHP, and JDBC.

## 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.

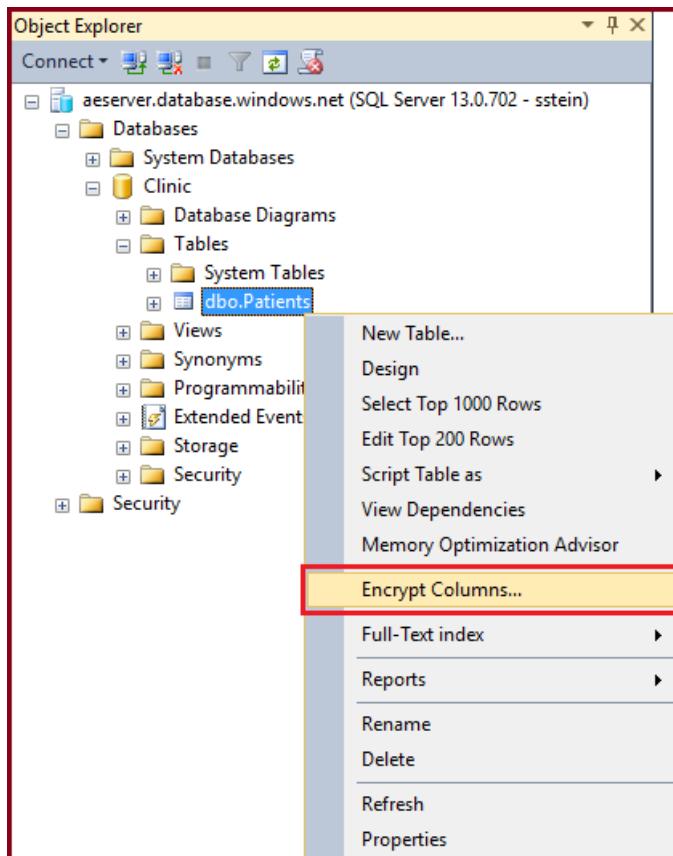
1. Expand **Databases**.
2. Right-click the **Clinic** database and click **New Query**.
3. Paste the following Transact-SQL (T-SQL) into the new query window and **Execute** it.

```
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:



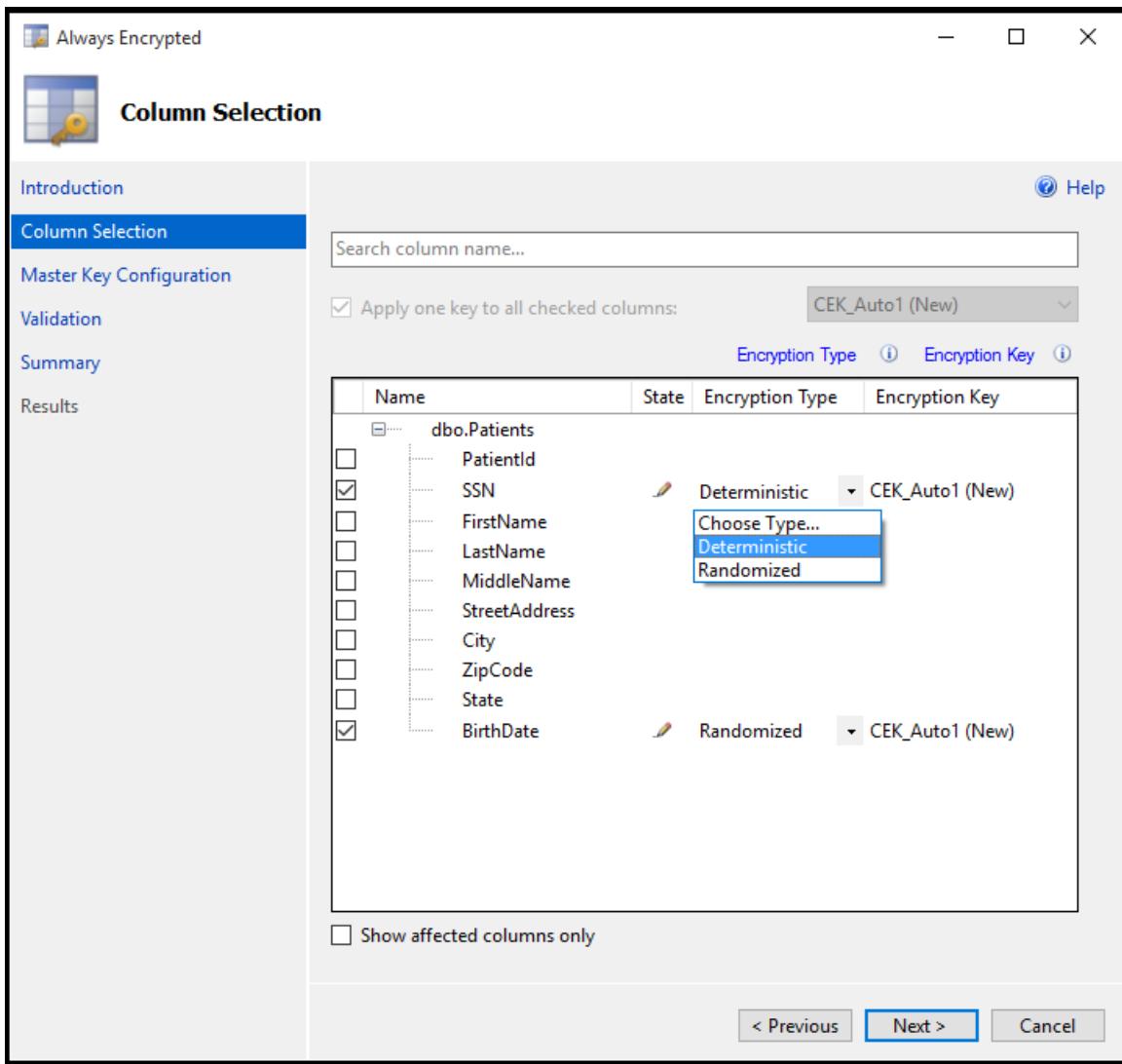
The Always Encrypted wizard includes the following sections: **Column Selection**, **Master Key Configuration**, **Validation**, and **Summary**.

#### 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**.

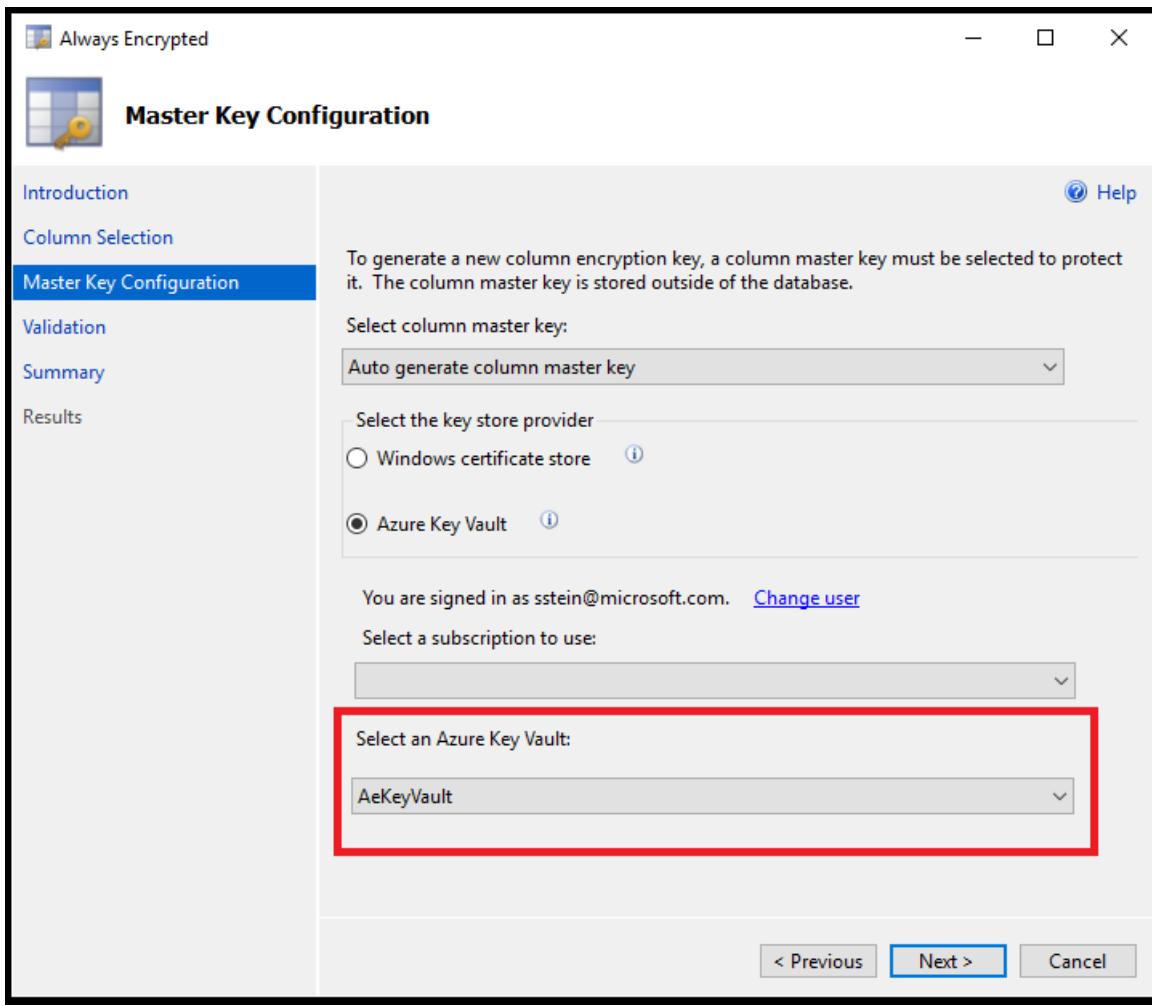


## Master Key Configuration

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**.

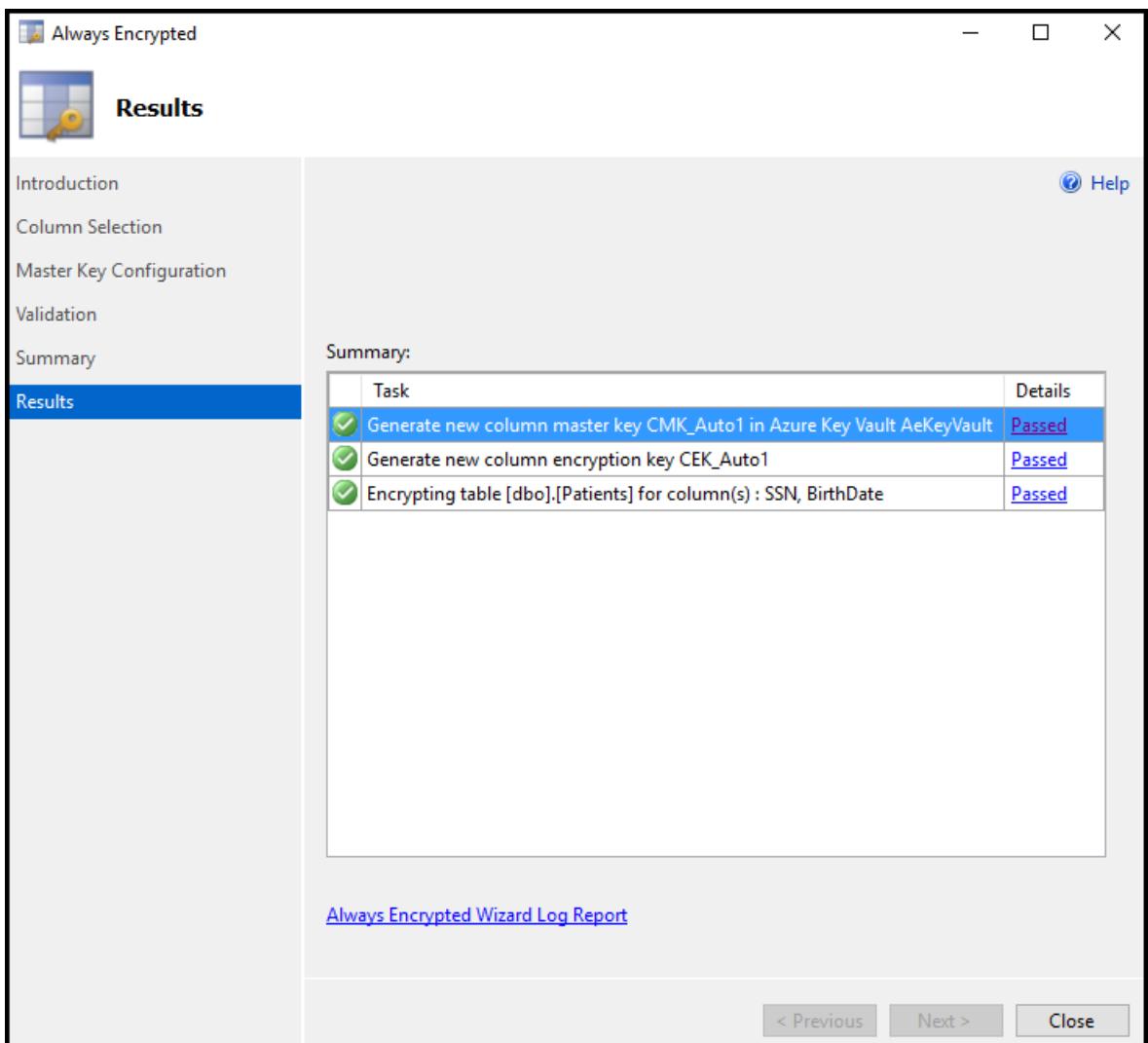


## Validation

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 = "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;
```

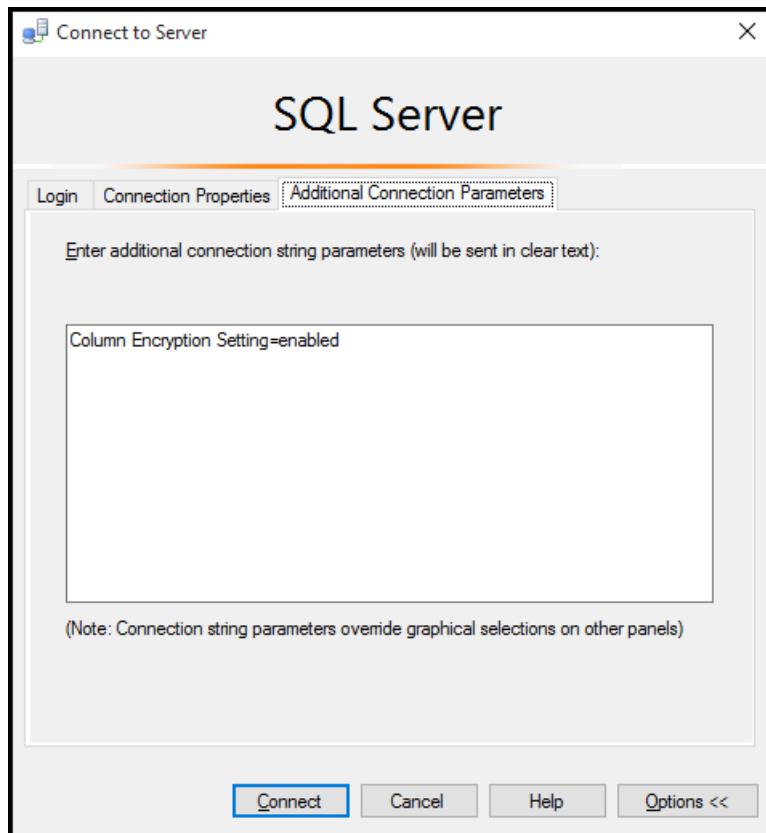
You can see that the encrypted columns do not contain any plaintext data.

|   | FirstName | LastName  | SSN  | BirthDate  |
|---|-----------|-----------|--|--|
| 1 | Orlando   | Gee       | 0x0195D36448C6B12BF988E9F9A5959384E238CF5F8B18344... | 0x01E574FA72017BF56013830652662116FFBBE84143DE2CD06... |
| 2 | Keith     | Harris    | 0x011B70A767292B4BE12E0C9B40038FDCA92613146DF1A20... | 0x011AF20C48A320D507D9E23405162F97B805226F1363ECE2...  |
| 3 | Donna     | Cameras   | 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**.



- 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      | Harrington | 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](#)

# Transparent data encryption for SQL Database and Data Warehouse

10/19/2018 • 7 minutes to read • [Edit Online](#)

Transparent data encryption (TDE) helps protect Azure SQL Database and Azure Data Warehouse against the threat of malicious activity. 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 will need to be manually enabled for older databases or Azure SQL Data Warehouse.

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.

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. 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. 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 share the same built-in certificate. Microsoft automatically rotates these certificates at least every 90 days.

Microsoft also seamlessly moves and manages the keys as needed for geo-replication and restores.

### IMPORTANT

All newly created SQL databases are encrypted by default by using service-managed transparent data encryption. Existing databases before May 2017 and databases created through restore, geo-replication, and database copy aren't encrypted by default.

## Bring Your Own Key

With Bring Your Own Key support, you can take control over your transparent data encryption keys and control who can access them and when. Key Vault, which is the Azure cloud-based external key management system, is the first key management service that transparent data encryption has integrated with Bring Your Own Key support.

With Bring Your Own Key support, the database encryption key is protected by an asymmetric key stored in Key Vault. The asymmetric key never leaves Key Vault. After the server has permissions to a key vault, the server sends basic key operation requests to it through Key Vault. You set the asymmetric key at the server level, and all databases under that server inherit it.

With Bring Your Own Key support, you now can control key management tasks such as key rotations and key vault permissions. You also can delete keys and enable auditing/reporting on all encryption keys. Key Vault provides central key management and uses tightly monitored hardware security modules. Key Vault promotes separation of management of keys and data to help meet regulatory compliance. To learn more about Key Vault, see the [Key Vault documentation page](#).

To learn more about transparent data encryption with Bring Your Own Key support for SQL Database and Data Warehouse, see [Transparent data encryption with Bring Your Own Key support](#).

To start using transparent data encryption with 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

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 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 set transparent data encryption 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.

The screenshot shows the Microsoft Azure portal interface for a SQL database named 'SampleDatabase'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Pricing tier (scale DTUs), Geo-Replication, Auditing & Threat Detection, Dynamic Data Masking, and Transparent data encryption (which is selected and highlighted). The main content area displays information about transparent data encryption, including a shield icon, a description of its purpose, a 'Learn more' link, a 'Data encryption' toggle switch set to 'ON', and an 'Encryption status' section indicating it is 'Encrypted'.

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, see the transparent data encryption settings under your server.

The screenshot shows the Microsoft Azure portal interface for a SQL server named 'SampleServer'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Firewall, Failover groups, Long-term backup retention, Auditing & Threat Detection, Transparent data encryption (selected), Active Directory admin, Deleted databases, Properties, Locks, Automation script, Automatic tuning, Recommendations, and New support request. The main content area displays information about transparent data encryption, including a shield icon, a description of its purpose, a 'Use your own key (preview)' section with a 'Yes' button, a 'Select a key' radio button, and a 'Key vault' dropdown set to 'ContosoDevKeyVault'. A tooltip provides information about permissions required for the key vault. To the right, a separate 'Key Picker' dialog box lists three keys: 'Key1 RSA', 'ContosoRSAKey0 RSA', and 'Key2 RSA', with 'Key1 RSA' selected.

## Manage transparent data encryption by using PowerShell

To configure transparent data encryption through PowerShell, you must be connected as the Azure Owner, Contributor, or SQL Security Manager.

| CMDLET   | DESCRIPTION  |
|--|--|
| <a href="#">Set-AzureRmSqlDatabaseTransparentDataEncryption</a>                | Enables or disables transparent data encryption for a database           |
| <a href="#">Get-Azure-Rm-Sql-Database-Transparent-Data-Encryption</a>          | Gets the transparent data encryption state for a database                |
| <a href="#">Get-Azure-Rm-Sql-Database-Transparent-Data-Encryption-Activity</a> | Checks the encryption progress for a database                            |
| <a href="#">Add-AzureRmSqlServerKeyVaultKey</a>                                | Adds a Key Vault key to a SQL Server instance                            |
| <a href="#">Get-AzureRmSqlServerKeyVaultKey</a>                                | Gets the Key Vault keys for a SQL server instance                        |
| <a href="#">Set-AzureRmSqlServerTransparentDataEncryptionProtector</a>         | Sets the transparent data encryption protector for a SQL Server instance |
| <a href="#">Get-AzureRmSqlServerTransparentDataEncryptionProtector</a>         | Gets the transparent data encryption protector                           |
| <a href="#">Remove-AzureRmSqlServerKeyVaultKey</a>                             | Removes a Key Vault key from a SQL Server instance                       |

## Manage transparent data encryption by using Transact-SQL

Connect to the database by using a login that is an administrator or member of the **dbmanager** role in the master database.

| COMMAND   | DESCRIPTION  |
|---|--|
| <a href="#">ALTER DATABASE (Azure SQL Database)</a>       | SET ENCRYPTION ON/OFF encrypts or decrypts a database  |
| <a href="#">sys.dm_database_encryption_keys</a>           | Returns information about the encryption state of a database and its associated database encryption keys               |
| <a href="#">sys.dm_pdw_nodes_database_encryption_keys</a> | Returns information about the encryption state of each data warehouse node and its associated database encryption keys |

You can't switch the transparent data encryption protector to a key from Key Vault by using Transact-SQL. Use PowerShell or the Azure portal.

## Manage transparent data encryption by using the REST API

To configure transparent data encryption through the REST API, you must be connected as the Azure Owner, Contributor, or SQL Security Manager.

| COMMAND                                     | DESCRIPTION  |
|---|--|
| <a href="#">Create Or Update Server</a>     | Adds an Azure Active Directory identity to a SQL Server instance (used to grant access to Key Vault) |
| <a href="#">Create Or Update Server Key</a> | Adds a Key Vault key to a SQL Server instance  |
| <a href="#">Delete Server Key</a>           | Removes a Key Vault key from a SQL Server instance   |

| COMMAND  | DESCRIPTION   |
|--|---|
| Get Server Keys  | Gets a specific Key Vault key from a SQL Server instance                  |
| List Server Keys By Server                                 | Gets the Key Vault keys for a SQL Server instance                         |
| Create Or Update Encryption Protector                      | Sets the transparent data encryption protector for a SQL Server instance  |
| Get Encryption Protector                                   | Gets the transparent data encryption protector for a SQL Server instance  |
| List Encryption Protectors By Server                       | Gets the transparent data encryption protectors for a SQL Server instance |
| Create Or Update Transparent Data Encryption Configuration | Enables or disables transparent data encryption for a database            |
| Get Transparent Data Encryption Configuration              | Gets the transparent data encryption configuration for a database         |
| List Transparent Data Encryption Configuration Results     | Gets the encryption result for a database                                 |

## Next steps

- For a general description of transparent data encryption, see [Transparent data encryption](#).
- To learn more about transparent data encryption with Bring Your Own Key support for SQL Database and Data Warehouse, see [Transparent data encryption with Bring Your Own Key support](#).
- To start using transparent data encryption with 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](#).
- For more information about Key Vault, see the [Key Vault documentation page](#).

# Azure SQL Transparent Data Encryption: Bring Your Own Key support

10/18/2018 • 14 minutes to read • [Edit Online](#)

Bring Your Own Key (BYOK) support for [Transparent Data Encryption \(TDE\)](#) allows you to encrypt the Database Encryption Key (DEK) with an asymmetric key called TDE Protector. The TDE Protector is stored under your control in [Azure Key Vault](#), Azure's cloud-based external key management system. Azure Key Vault is the first key management service with which TDE has integrated support for BYOK. The TDE DEK, which is stored on the boot page of a database is encrypted and decrypted by the TDE protector. The TDE Protector is stored in Azure Key Vault and never leaves the key vault. If the server's access to the key vault is revoked, a database cannot be decrypted and read into memory. The TDE protector is set at the logical server level and is inherited by all databases associated with that server.

With BYOK support, users can now control key management tasks including key rotations, key vault permissions, deleting keys, 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 regulatory compliance.

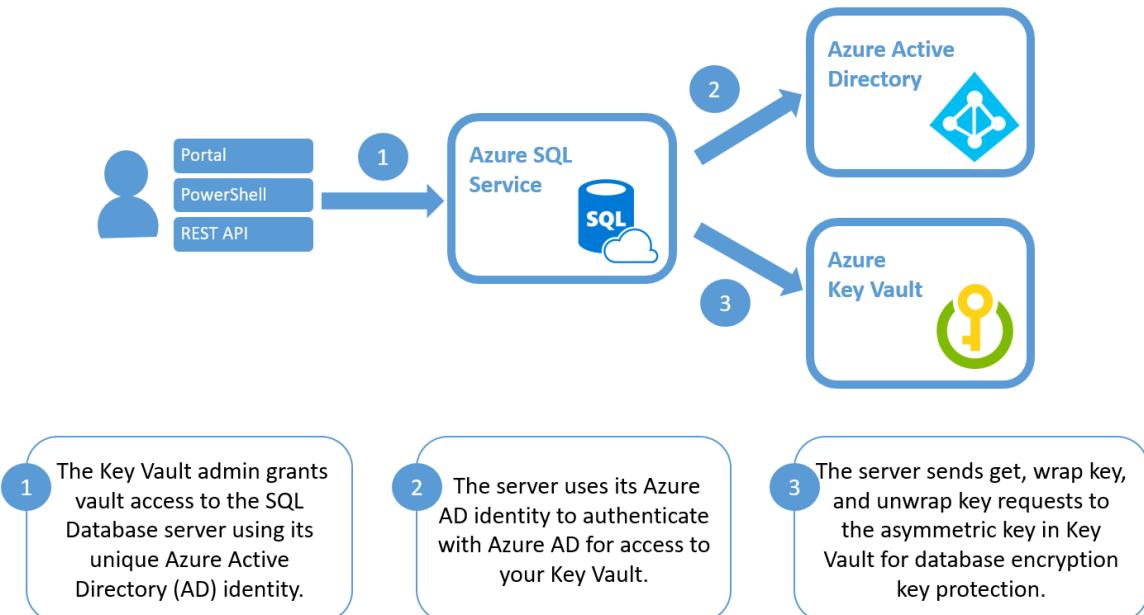
TDE with BYOK provides the following benefits:

- Increased transparency and granular control with the ability to self-manage the TDE protector
- Central management of TDE protectors (along with other keys and secrets used in other Azure services) by hosting them in Key Vault
- Separation of key and data management responsibilities within the organization, to support separation of duties
- Greater trust from your own clients, since Key Vault is designed so that Microsoft does not see or extract any encryption keys.
- Support for key rotation

## IMPORTANT

For those using service-managed TDE who would like to start using Key Vault, TDE remains enabled during the process of switching over to a TDE protector in Key Vault. There is no downtime nor re-encryption of the database files. Switching from a service-managed key to a Key Vault key only requires re-encryption of the database encryption key (DEK), which is a fast and online operation.

## How does TDE with BYOK support work



When TDE is first configured to use a TDE protector from Key Vault, the server sends the DEK of each TDE-enabled database to Key Vault for a wrap key request. Key Vault returns the encrypted database encryption key, which is stored in the user database.

#### IMPORTANT

It is important to note that **once a TDE Protector is stored in Azure Key Vault, it never leaves the Azure Key Vault**. The logical server can only send key operation requests to the TDE protector key material within Key Vault, and **never accesses or caches the TDE protector**. The Key Vault administrator has the right to revoke Key Vault permissions of the server at any point, in which case all connections to the server are cut off.

## Guidelines for configuring TDE with BYOK

### General Guidelines

- Ensure Azure Key Vault and Azure SQL Database are going to be in the same tenant. Cross-tenant key vault and server interactions **are not supported**.
- Decide which subscriptions are going to be used for the required resources – moving the server across subscriptions later requires a new setup of TDE with BYOKs. Learn more about [moving resources](#)
- When configuring TDE with BYOK, it is important to consider the load placed on the key vault by repeated wrap/unwrap operations. For example, since all databases associated with a logical server use the same TDE protector, a failover of that server will trigger as many key operations against the vault as there are databases in the server. Based on our experience and documented [key vault service limits](#), we recommend associating at most 500 Standard / General Purpose or 200 Premium / Business Critical databases with one Azure Key Vault in a single subscription to ensure consistently high availability when accessing the TDE protector in the vault.
- Recommended: Keep a copy of the TDE protector on premises. This requires an HSM device to create a TDE Protector locally and a key escrow system to store a local copy of the TDE Protector. Learn [how to transfer a key from a local HSM to Azure Key Vault](#).

### Guidelines for configuring Azure Key Vault

- Create a key vault with [soft-delete](#) enabled to protect from data loss in case of accidental key – or key vault – deletion. You must use [PowerShell to enable the "soft-delete" property](#) on the key vault (this option is not available from the AKV Portal yet – but required by SQL):
  - Soft deleted resources are retained for a set period of time, 90 days unless they are recovered or

purged.

- The **recover** and **purge** actions have their own permissions associated in a key vault access policy.
- Set a resource lock on the key vault to control who can delete this critical resource and help to prevent accidental or unauthorized deletion. [Learn more about resource locks](#)
- Grant the logical server access to the key vault using its Azure Active Directory (Azure AD) Identity. When using the Portal UI, the Azure AD identity gets automatically created and the key vault access permissions are granted to the server. Using PowerShell to configure TDE with BYOK, the Azure AD identity must be created and completion should be verified. See [Configure TDE with BYOK](#) for detailed step-by-step instructions when using PowerShell.

**NOTE**

If the Azure AD Identity **is accidentally deleted or the server's permissions are revoked** using the key vault's access policy, the server loses access to the key vault, and TDE encrypted databases are dropped within 24 hours.

- When using firewalls and virtual networks with Azure Key Vault, you must configure the following:
  - Allow trusted Microsoft services to bypass this firewall – chose YES

**NOTE**

If TDE encrypted SQL databases lose access to the key vault because they cannot bypass the firewall, the databases are dropped within 24 hours.

- Enable auditing and reporting on all encryption keys: Key Vault provides logs that are easy to inject into other security information and event management (SIEM) tools. Operations Management Suite (OMS) [Log Analytics](#) is one example of a service that is already integrated.
- To ensure high-availability of encrypted databases, configure each logical server with two Azure Key Vaults that reside in different regions.

#### Guidelines for configuring the TDE Protector (asymmetric key)

- Create your encryption key locally on a local HSM device. Ensure this is an asymmetric, RSA 2048 key so it is storable in Azure Key Vault.
- Escrow the key in a key escrow system.
- Import the encryption key file (.pfx, .byok, or .backup) to Azure Key Vault.

**NOTE**

For testing purposes, it is possible to create a key with Azure Key Vault, however this key cannot be escrowed, because the private key can never leave the key vault. Always back up and escrow keys used to encrypt production data, as the loss of the key (accidental deletion in key vault, expiration etc.) results in permanent data loss.

- Use a key without an expiration date – and never set an expiration date on a key already in use: **once the key expires, the encrypted databases lose access to their TDE Protector and are dropped within 24 hours.**
- Ensure the key is enabled and has permissions to perform *get*, *wrap key*, and *unwrap key* operations.
- Create an Azure Key Vault key backup before using the key in Azure Key Vault for the first time. Learn more about the [Backup-AzureKeyVaultKey](#) command.
- Create a new backup whenever any changes are made to the key (for example, add ACLs, add tags, add key attributes).

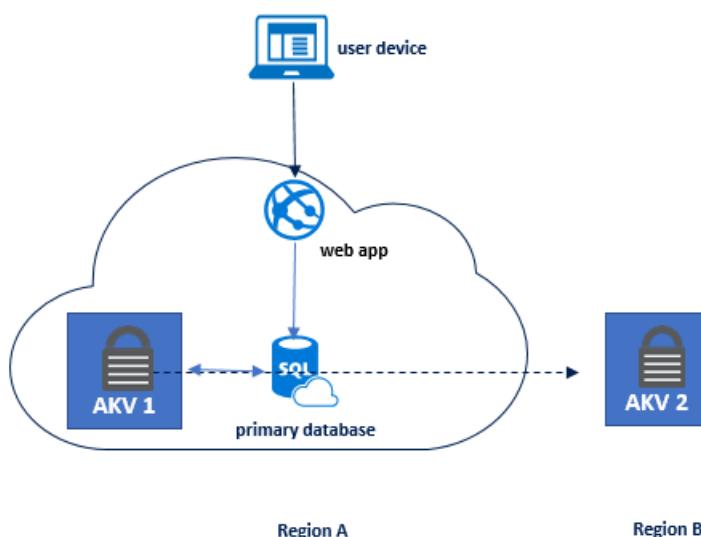
- Keep previous versions of the key in the key vault when rotating keys, so older database backups can be restored. When the TDE Protector is changed for a database, old backups of the database **are not updated** to use the latest TDE Protector. Each backup needs the TDE Protector it was created with at restore time. Key rotations can be performed following the instructions at [Rotate the Transparent Data Encryption Protector Using PowerShell](#).
- Keep all previously used keys in Azure Key Vault after changing back to service-managed keys. This ensures database backups can be restored with the TDE protectors stored in Azure Key Vault. TDE protectors created with Azure Key Vault have to be maintained until all stored backups have been created with service-managed keys.
- Make recoverable backup copies of these keys using [Backup-AzureKeyVaultKey](#).
- To remove a potentially compromised key during a security incident without the risk of data loss, follow the steps at [Remove a potentially compromised key](#).

## High Availability, Geo-Replication, and Backup / Restore

### High availability and disaster recovery

How to configure high availability with Azure Key Vault depends on the configuration of your database and logical server, and here are the recommended configurations for two distinct cases. The first case is a stand-alone database or logical server with no configured geo redundancy. The second case is a database or logical server configured with failover groups or geo-redundancy, where it must be ensured that each geo-redundant copy has a local Azure Key Vault within the failover group to ensure geo-failovers work.

In the first case, if you require high availability of a database and logical server with no configured geo-redundancy, it is highly recommended to configure the server to use two different key vaults in two different regions with the same key material. This can be accomplished by creating a TDE protector using the primary Key Vault co-located in the same region as the logical 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-AzureKeyVaultKey` cmdlet to retrieve the key in encrypted format from the primary key vault and then use the `Restore-AzureKeyVaultKey` cmdlet and specify a key vault in the second region.



## How to configure Geo-DR with Azure Key Vault

To maintain high availability of TDE Protectors for encrypted databases, it is required to configure redundant Azure Key Vaults based on the existing or desired SQL Database failover groups or active geo-replication

instances. Each geo-replicated server requires a separate key vault, that must be co-located with the server in the same Azure region. Should a primary database become inaccessible due to an outage in one region and a failover is triggered, the secondary database is able to take over using the secondary key vault.

For Geo-Replicated Azure SQL databases, the following Azure Key Vault configuration is required:

- One primary database with a key vault in region and one secondary database with a key vault in region.
- At least one secondary is required, up to four secondaries are supported.
- Secondaries of secondaries (chaining) are not supported.

The following section will go over the setup and configuration steps in more detail.

### Azure Key Vault Configuration Steps

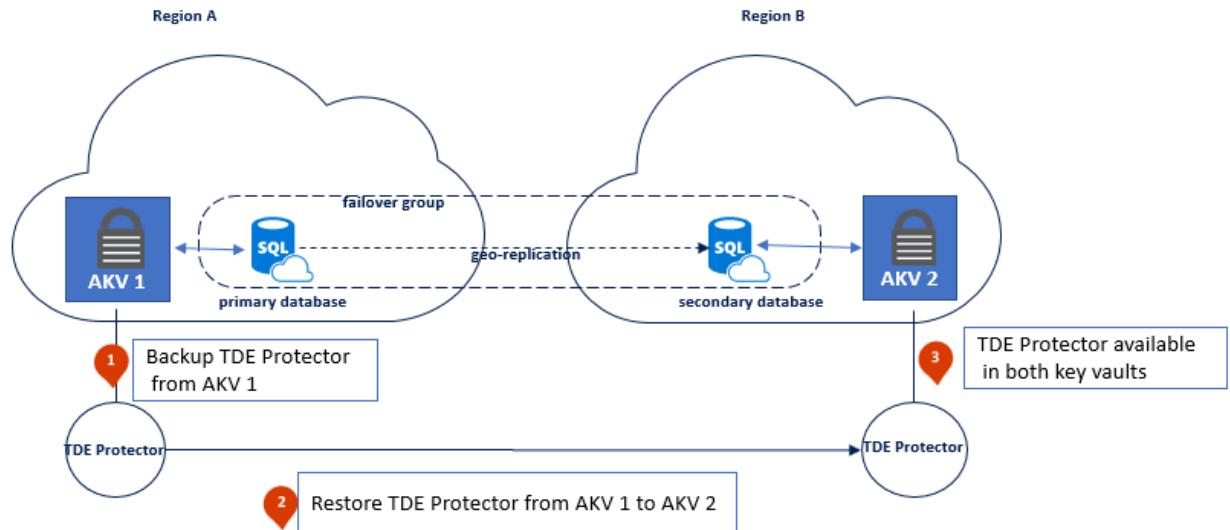
- Install [PowerShell](#)
- Create two Azure Key Vaults in two different regions using [PowerShell to enable the "soft-delete" property](#) on the key vaults (this option is not available from the AKV Portal yet – but required by SQL).
- Both Azure Key Vaults must be located in the two regions available in the same Azure Geo in order for backup and restore of keys to work. If you need the two key vaults to be located in different geos to meet SQL Geo-DR requirements, follow the [BYOK Process](#) that allows keys to be imported from an on-premises HSM.
- Create a new key in the first key vault:
  - RSA/RSA-HSA 2048 key
  - No expiration dates
  - Key is enabled and has permissions to perform get, wrap key, and unwrap key operations
- Back up the primary key and restore the key to the second key vault. See [BackupAzureKeyVaultKey](#) and [Restore-AzureKeyVaultKey](#).

### Azure SQL Database Configuration Steps

The following configuration steps differ whether starting with a new SQL deployment or if working with an already existing SQL Geo-DR deployment. We outline the configuration steps for a new deployment first, and then explain how to assign TDE Protectors stored in Azure Key Vault to an existing deployment that already has a Geo-DR link established.

#### Steps for a new deployment:

- Create the two logical SQL servers in the same two regions as the previously created key vaults.
- Select the logical server TDE pane, and for each logical SQL server:
  - Select the AKV in the same region
  - Select the key to use as TDE Protector – each server will use the local copy of the TDE Protector.
  - Doing this in the Portal will create an [AppID](#) for the logical SQL server, which is used to assign the logical SQL Server permissions to access the key vault – do not delete this identity. Access can be revoked by removing the permissions in Azure Key Vault instead for the logical SQL server, which is used to assign the logical SQL Server permissions to access the key vault.
- Create the primary database.
- Follow the [active geo-replication guidance](#) to complete the scenario, this step will create the secondary database.

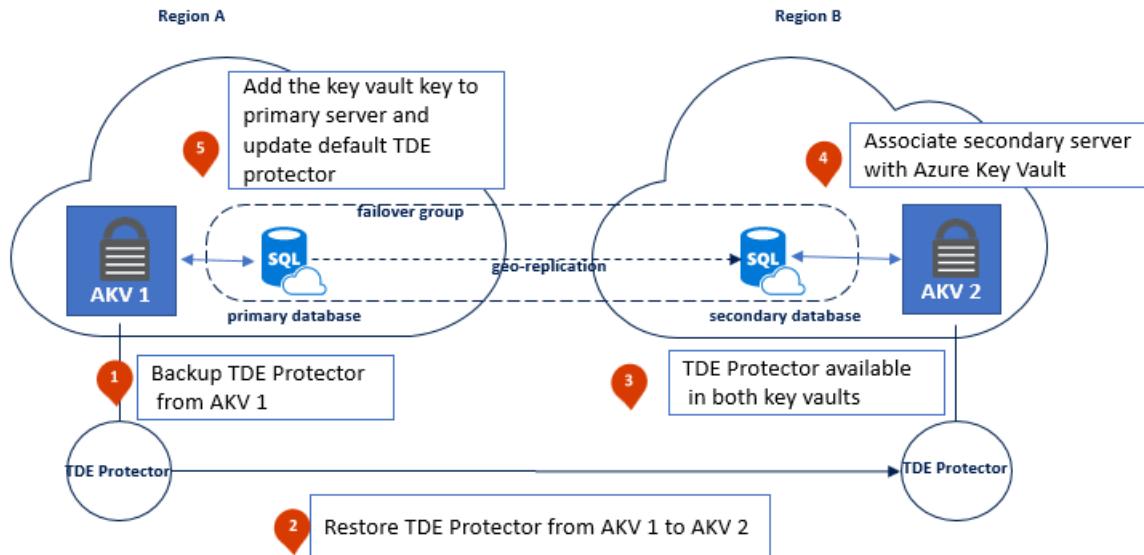
**NOTE**

It is important to ensure that the same TDE Protectors are present in both key vaults, before proceeding to establish the geo-link between the databases.

**Steps for an existing SQL DB with Geo-DR deployment:**

Because the logical SQL servers already exist, and primary and secondary databases are already assigned, the steps to configure Azure Key Vault must be performed in the following order:

- Start with the logical SQL Server that hosts the secondary database:
  - Assign the key vault located in the same region
  - Assign the TDE Protector
- Now go to the logical SQL Server that hosts the primary database:
  - Select the same TDE Protector as used for the secondary DB



#### NOTE

When assigning the key vault to the server, it is important to start with the secondary server. In the second step assign the key vault to the primary server and update the TDE Protector, the Geo-DR link will continue to work because at this point the TDE Protector used by the replicated database is available to both servers.

Before enabling TDE with customer managed keys in Azure Key Vault for a SQL Database Geo-DR scenario, it is important to create and maintain two Azure Key Vaults with identical contents in the same regions that will be used for SQL Database geo-replication. "Identical contents" specifically means that both key vaults must contain copies of the same TDE Protector(s) so that both servers have access to the TDE Protectors used by all databases. Going forward, it is required to keep both key vaults in sync, which means they must contain the same copies of TDE Protectors after key rotation, maintain old versions of keys used for log files or backups, TDE Protectors must maintain the same key properties and the key vaults must maintain the same access permissions for SQL.

Follow the steps in [Active geo-replication overview](#) to test and trigger a failover, which should be done on a regular basis to confirm the access permissions for SQL to both key vaults have been maintained.

#### Backup and Restore

Once a database is encrypted with TDE using a key from Key Vault, any generated backups are also encrypted with the same TDE Protector.

To restore a backup encrypted with a TDE Protector from Key Vault, make sure that the key material is still in the original vault under the original key name. When the TDE Protector is changed for a database, old backups of the database **are not** updated to use the latest TDE Protector. Therefore, we recommend that you keep all old versions of the TDE Protector in Key Vault, so database backups can be restored.

If a key that might be needed for restoring a backup is no longer in its original key vault, the following error message is returned: "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 this, run the [Get-AzureRmSqlServerKeyVaultKey](#) cmdlet to return the list of keys from Key Vault that were added to the server (unless they were deleted by a user). To ensure all backups can be restored, make sure the target server for the backup has access to all of these keys.

```
Get-AzureRmSqlServerKeyVaultKey ` 
-ServerName <LogicalServerName> ` 
-ResourceGroup <SQLDatabaseResourceGroupName>
```

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](#).

Additional consideration for backed up log files: Backed up log files remain encrypted with the original TDE Encryptor, even if the TDE Protector 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.

# PowerShell and CLI: Enable Transparent Data Encryption using your own key from Azure Key Vault

9/25/2018 • 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 Bring Your Own Key (BYOK) Support, visit [TDE Bring Your Own Key to Azure SQL](#).

## 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 version 4.2.0 or newer installed and running.
- Create an Azure Key Vault and Key to use for TDE.
  - [PowerShell instructions from Key Vault](#)
  - [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](#)
  - [How to use Key Vault soft-delete with PowerShell](#)
- 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

## Step 1. 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-AzureRmSqlServer `  
-ResourceGroupName <SQLDatabaseResourceGroupName> `  
-ServerName <LogicalServerName> `  
-AssignIdentity
```

If you are creating a server, use the [New-AzureRmSqlServer](#) cmdlet with the tag `-Identity` to add an Azure AD identity during server creation:

```
$server = New-AzureRmSqlServer `  
-ResourceGroupName <SQLDatabaseResourceGroupName> `  
-Location <RegionName> `  
-ServerName <LogicalServerName> `  
-ServerVersion "12.0" `  
-SqlAdministratorCredentials <PSCredential> `  
-AssignIdentity
```

## Step 2. Grant Key Vault permissions to your server

Use the [Set-AzureRmKeyVaultAccessPolicy](#) cmdlet to grant your server access to the key vault before using a key

from it for TDE.

```
Set-AzureRmKeyVaultAccessPolicy ` 
-VaultName <KeyVaultName> ` 
-ObjectId $server.Identity.PrincipalId ` 
-PermissionsToKeys get, wrapKey, unwrapKey
```

## Step 3. Add the Key Vault key to the server and set the TDE Protector

- Use the [Add-AzureRmSqlServerKeyVaultKey](#) cmdlet to add the key from the Key Vault to the server.
- Use the [Set-AzureRmSqlServerTransparentDataEncryptionProtector](#) cmdlet to set the key as the TDE protector for all server resources.
- Use the [Get-AzureRmSqlServerTransparentDataEncryptionProtector](#) 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-AzureRmSqlServerKeyVaultKey ` 
-ResourceGroupName <SQLDatabaseResourceGroupName> ` 
-ServerName <LogicalServerName> ` 
-KeyId <KeyVaultKeyId>

<# Set the key as the TDE protector for all resources under the server #>
Set-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-ResourceGroupName <SQLDatabaseResourceGroupName> ` 
-ServerName <LogicalServerName> ` 
-Type AzureKeyVault ` 
-KeyId <KeyVaultKeyId>

<# To confirm that the TDE protector was configured as intended: #>
Get-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-ResourceGroupName <SQLDatabaseResourceGroupName> ` 
-ServerName <LogicalServerName>
```

## Step 4. Turn on TDE

Use the [Set-AzureRMSqlDatabaseTransparentDataEncryption](#) cmdlet to turn on TDE.

```
Set-AzureRMSqlDatabaseTransparentDataEncryption ` 
-ResourceGroupName <SQLDatabaseResourceGroupName> ` 
-ServerName <LogicalServerName> ` 
-DatabaseName <DatabaseName> ` 
-State "Enabled"
```

Now the database or data warehouse has TDE enabled with an encryption key in Key Vault.

## Step 5. Check the encryption state and encryption activity

Use the [Get-AzureRMSqlDatabaseTransparentDataEncryption](#) to get the encryption state and the [Get-AzureRMSqlDatabaseTransparentDataEncryptionActivity](#) to check the encryption progress for a database or data warehouse.

```
# Get the encryption state  
Get-AzureRMSqlDatabaseTransparentDataEncryption `  
-ResourceGroupName <SQLDatabaseResourceGroupName> `  
-ServerName <LogicalServerName> `  
-DatabaseName <DatabaseName> `  
  
<# Check the encryption progress for a database or data warehouse #>  
Get-AzureRMSqlDatabaseTransparentDataEncryptionActivity `  
-ResourceGroupName <SQLDatabaseResourceGroupName> `  
-ServerName <LogicalServerName> `  
-DatabaseName <DatabaseName>
```

## Other useful PowerShell cmdlets

- Use the [Set-AzureRMSqlDatabaseTransparentDataEncryption](#) cmdlet to turn off TDE.

```
Set-AzureRMSqlDatabaseTransparentDataEncryption `  
-ServerName <LogicalServerName> `  
-ResourceGroupName <SQLDatabaseResourceGroupName> `  
-DatabaseName <DatabaseName> `  
-State "Disabled"
```

- Use the [Get-AzureRmSqlServerKeyVaultKey](#) 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-AzureRmSqlServerKeyVaultKey `  
-ServerName <LogicalServerName> `  
-ResourceGroupName <SQLDatabaseResourceGroupName>
```

- Use the [Remove-AzureRmSqlServerKeyVaultKey](#) to remove a Key Vault key from the server.

```
<# The key set as the TDE Protector cannot be removed. #>  
Remove-AzureRmSqlServerKeyVaultKey `  
-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 using the [Get-AzureRmSubscription](#) cmdlet.

```
Get-AzureRmSubscription `  
-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](#).

## Prerequisites for CLI

- 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.
- Command-Line Interface version 2.0 or later. To install the latest version and connect to your Azure subscription, see [Install and Configure the Azure Cross-Platform Command-Line Interface 2.0](#).
- Create an Azure Key Vault and Key to use for TDE.
  - [Manage Key Vault using CLI 2.0](#)
  - [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](#)
    - [How to use Key Vault soft-delete with CLI](#)
- 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

## Step 1. Create a server and assign an Azure AD identity to your server

```
cli
# create server (with identity) and database
az sql server create -n "ServerName" -g "ResourceGroupName" -l "westus" -u "cloudsa" -p
"YourFavoritePassWord99@34" -I
az sql db create -n "DatabaseName" -g "ResourceGroupName" -s "ServerName"
```

## Step 2. Grant Key Vault permissions to your server

```
cli
# create key vault, key and grant permission
az keyvault create -n "VaultName" -g "ResourceGroupName"
az keyvault key create -n myKey -p software --vault-name "VaultName"
az keyvault set-policy -n "VaultName" --object-id "ServerIdentityObjectId" -g "ResourceGroupName" --key-
permissions wrapKey unwrapKey get list
```

## Step 3. Add the Key Vault key to the server and set the TDE Protector

```
cli
# add server key and update encryption protector
az sql server key create -g "ResourceGroupName" -s "ServerName" -t "AzureKeyVault" -u "FullVersionedKeyUri"
az sql server tde-key update -g "ResourceGroupName" -s "ServerName" -t AzureKeyVault -u
"FullVersionedKeyUri"
```

#### 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>

## Step 4. Turn on TDE

```
cli
# enable encryption
az sql db tde create -n "DatabaseName" -g "ResourceGroupName" -s "ServerName" --status Enabled
```

Now the database or data warehouse has TDE enabled with an encryption key in Key Vault.

## Step 5. Check the encryption state and encryption activity

```
cli
# get encryption scan progress
az sql db tde show-activity -n "DatabaseName" -g "ResourceGroupName" -s "ServerName"

# get whether encryption is on or off
az sql db tde show-configuration -n "DatabaseName" -g "ResourceGroupName" -s "ServerName"
```

## SQL CLI References

<https://docs.microsoft.com/cli/azure/sql?view=azure-cli-latest>

<https://docs.microsoft.com/cli/azure/sql/server/key?view=azure-cli-latest>

<https://docs.microsoft.com/cli/azure/sql/server/tde-key?view=azure-cli-latest>

<https://docs.microsoft.com/cli/azure/sql/db/tde?view=azure-cli-latest>

# Rotate the Transparent Data Encryption (TDE) protector using PowerShell

10/12/2018 • 2 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 version 3.7.0 or newer 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.

## Option 1: Auto rotation

Generate a new version of the existing TDE protector key in Key Vault, under the same key name and key vault. The Azure SQL service starts using this new version within 24 hours.

To create a new version of the TDE protector using the [Add-AzureKeyVaultKey](#) cmdlet:

```
Add-AzureKeyVaultKey ` 
-VaultName <KeyVaultName> ` 
-Name <KeyVaultKeyName> ` 
-Destination <HardwareOrSoftware>
```

## Option 2: Manual rotation

The option uses the [Add-AzureKeyVaultKey](#), [Add-AzureRmSqlServerKeyVaultKey](#), and [Set-AzureRmSqlServerTransparentDataEncryptionProtector](#) cmdlets to add a completely new key, which could be under a new key name or even another key vault.

#### NOTE

The combined length for the key vault name and key name cannot exceed 94 characters.

```
# Add a new key to Key Vault
Add-AzureKeyVaultKey ` 
-VaultName <KeyVaultName> ` 
-Name <KeyVaultKeyName> ` 
-Destination <HardwareOrSoftware>

# Add the new key from Key Vault to the server
Add-AzureRmSqlServerKeyVaultKey ` 
-KeyId <KeyVaultKeyId> ` 
-ServerName <LogicalServerName> ` 
-ResourceGroup <SQLDatabaseResourceGroupName>

<# Set the key as the TDE protector for all resources
under the server #>
Set-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-Type AzureKeyVault ` 
-KeyId <KeyVaultKeyId> ` 
-ServerName <LogicalServerName> ` 
-ResourceGroup <SQLDatabaseResourceGroupName>
```

## Other useful PowerShell cmdlets

- To switch the TDE protector from Microsoft-managed to BYOK mode, use the [Set-AzureRmSqlServerTransparentDataEncryptionProtector](#) cmdlet.

```
Set-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-Type AzureKeyVault ` 
-KeyId <KeyVaultKeyId> ` 
-ServerName <LogicalServerName> ` 
-ResourceGroup <SQLDatabaseResourceGroupName>
```

- To switch the TDE protector from BYOK mode to Microsoft-managed, use the [Set-AzureRmSqlServerTransparentDataEncryptionProtector](#) cmdlet.

```
Set-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-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 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

10/12/2018 • 3 minutes to read • [Edit Online](#)

## Prerequisites

- You must have an Azure subscription and be an administrator on that subscription
- You must have Azure PowerShell version 4.2.0 or newer 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.

## 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 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 can result in **data loss**.

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, **all connections to the encrypted databases under the server are blocked, and these databases go offline and get dropped within 24 hours**. Old backups encrypted with the compromised key are no longer accessible.

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

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-AzureRmSqlServerKeyVaultKey](#) and [Set-AzureRmSqlServerTransparentDataEncryptionProtector](#) cmdlets and update it as the server's new TDE protector.

```
# Add the key from Key Vault to the server
Add-AzureRmSqlServerKeyVaultKey ` 
-ResourceGroupName <SQLDatabaseResourceGroupName> ` 
-ServerName <LogicalServerName> ` 
-KeyId <KeyVaultKeyId>

# Set the key as the TDE protector for all resources under the server
Set-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-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-AzureRmSqlServerTransparentDataEncryptionProtector](#) 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-AzureRmSqlServerTransparentDataEncryptionProtector ` 
-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-AzureKeyVaultKey ` 
-VaultName <KeyVaultName> ` 
-Name <KeyVaultKeyName> ` 
-OutputFile <DesiredBackupFilePath>
```

5. Delete the compromised key from Key Vault using the [Remove-AzureKeyVaultKey](#) cmdlet.

```
Remove-AzureKeyVaultKey ` 
-VaultName <KeyVaultName> ` 
-Name <KeyVaultKeyName>
```

6. To restore a key to Key Vault in the future using the [Restore-AzureKeyVaultKey](#) cmdlet:

```
Restore-AzureKeyVaultKey ` 
-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

## 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](#)

# Dynamically scale database resources with minimal downtime

10/8/2018 • 4 minutes to read • [Edit Online](#)

Azure SQL Database enables you to dynamically add more resources to your database with minimal downtime.

## 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 of memory, and the amount and speed of storage. This purchasing model offers three service tiers: General Purpose, Business Critical, and Hyperscale (preview).

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.

### IMPORTANT

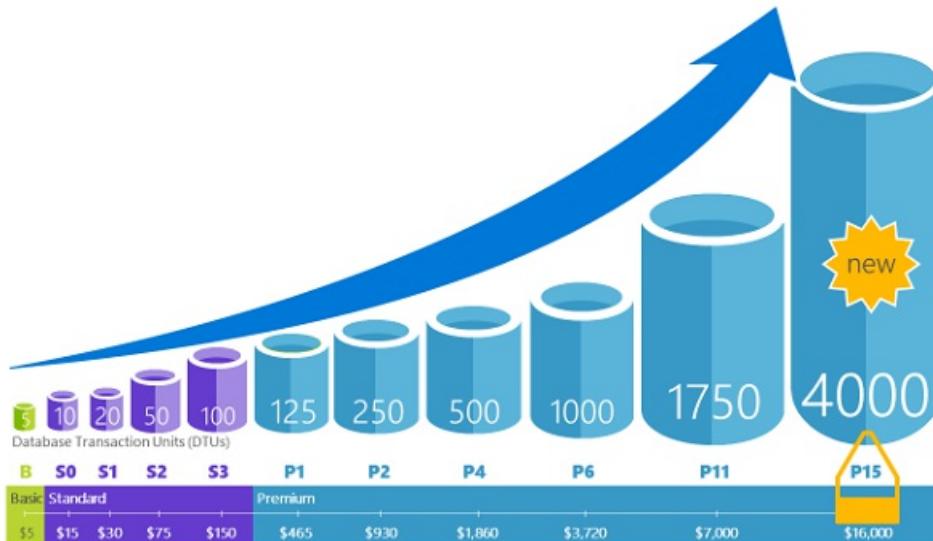
Hyperscale service tier is currently in public preview and available in limited Azure regions. You can't update a Hyperscale database to other service tiers. For test purpose, we recommend you make a copy of your current database, and update the copy to Hyperscale service tier.

### NOTE

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.

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:

- In [Azure SQL Single Database](#), you can use either [DTU](#) or [vCore](#) models to define maximum amount of resources that will be assigned to each database.
- [Azure SQL 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.
- [Azure SQL Elastic Pools](#) enable you to define maximum resource limit per group of databases in the pool.

## Alternative scale methods

Scaling resources is the easiest and the most effective way to improve performance of your database without changing either database or application code. In some cases, even the highest service tiers, compute sizes, and performance optimizations might not handle your workload on successful and cost-effective way. In that cases you have these additional options to scale your database:

- [Read scale-out](#) is a feature available in where you are getting one read-only replica of your data where you can execute demanding read-only queries such as reports. Red-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](#).

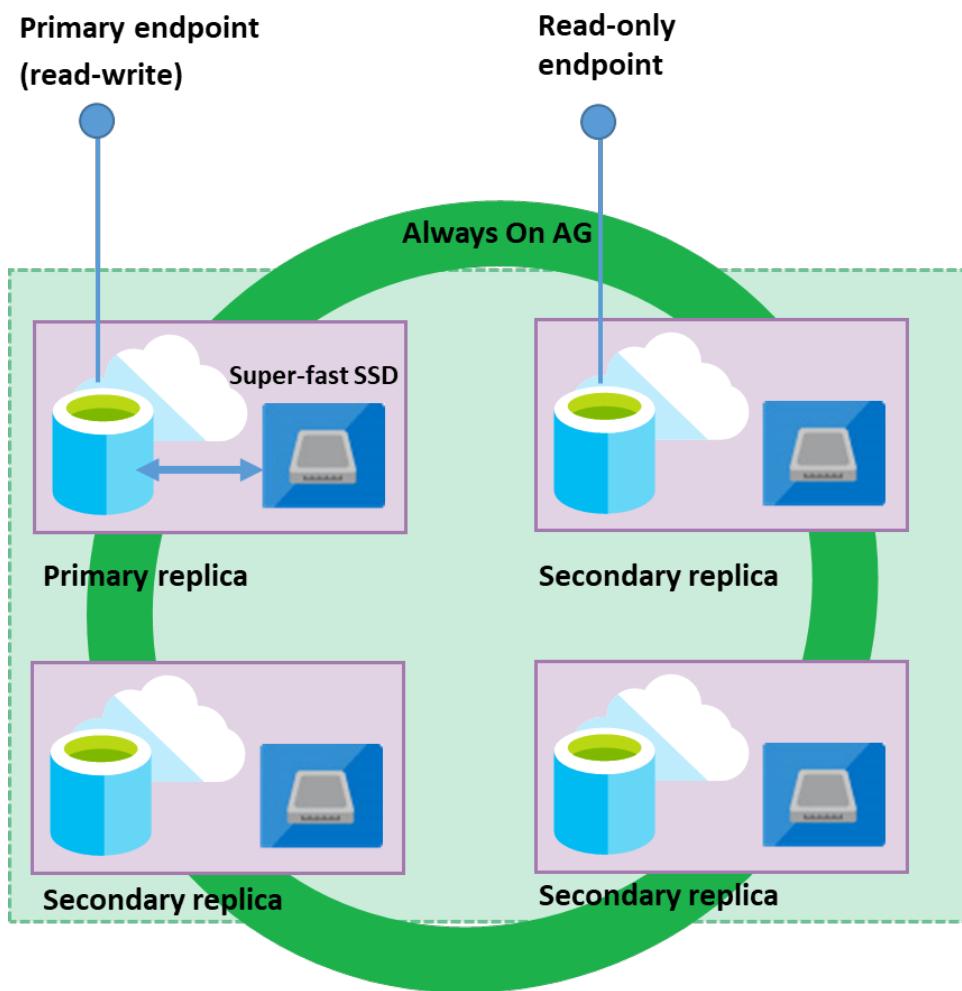
# Use read-only replicas to load balance read-only query workloads (preview)

10/19/2018 • 5 minutes to read • [Edit Online](#)

**Read Scale-Out** allows you to load balance Azure SQL Database read-only workloads using the capacity of one read-only replica.

## Overview of Read Scale-Out

Each database in the Premium tier ([DTU-based purchasing model](#)) or in the Business Critical tier ([vCore-based purchasing model](#)) is automatically provisioned with several AlwaysON replicas to support the availability SLA.



## Business Critical service tier: collocated compute and storage

These replicas are provisioned with the same compute size as the read-write replica used by the regular database connections. 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, and therefore could gain performance benefits using this additional capacity at no extra cost.

To use the Read Scale-Out feature with a particular database, you must explicitly enable it when creating the database or afterwards by altering its configuration using PowerShell by invoking the [Set-AzureRmSqlDatabase](#) or the [New-AzureRmSqlDatabase](#) cmdlets or through the Azure Resource Manager REST API using the [Databases - Create or Update](#) method.

After Read Scale-Out is enabled for a database, applications connecting to that database will be directed to either the read-write replica or to a read-only replica of that database according to the `ApplicationIntent` property configured in the application's connection string. For information on the `ApplicationIntent` property, see [Specifying Application Intent](#).

If Read Scale-Out is disabled or you set the `ReadScale` property in an unsupported service tier, all connections are directed to the read-write replica, independent of the `ApplicationIntent` property.

**NOTE**

During preview, Query Data Store and Extended Events 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 can 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. This is because the transaction log redo to the replicas is asynchronous.

**NOTE**

Replication latencies within the region are low and this situation is rare.

## Connecting 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;
```

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.

## Enable and disable Read Scale-Out

Read Scale-Out is enabled by default in [Managed Instance](#) Business Critical tier(Preview). It should be explicitly enabled in [database placed on logical server](#) Premium and Business Critical tiers. The methods for enabling and disabling Read Scale-Out is described here.

### Enable and disable Read Scale-Out using Azure PowerShell

Managing Read Scale-Out in Azure PowerShell requires the December 2016 Azure PowerShell release or newer. For the newest PowerShell release, see [Azure PowerShell](#).

Enable or disable read scale-out in Azure PowerShell by invoking the [Set-AzureRmSqlDatabase](#) cmdlet and passing in the desired value – `Enabled` or `Disabled` -- for the `-ReadScale` parameter. Alternatively, you may use the [New-AzureRmSqlDatabase](#) cmdlet to create a new database with read scale-out enabled.

For example, to enable read scale-out for an existing database (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
Set-AzureRmSqlDatabase -ResourceGroupName <myresourcegroup> -ServerName <myserver> -DatabaseName <mydatabase>  
-ReadScale Enabled
```

To disable read scale-out for an existing database (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
Set-AzureRmSqlDatabase -ResourceGroupName <myresourcegroup> -ServerName <myserver> -DatabaseName <mydatabase>  
-ReadScale Disabled
```

To create a new database with read scale-out enabled (replacing the items in the angle brackets with the correct values for your environment and dropping the angle brackets):

```
New-AzureRmSqlDatabase -ResourceGroupName <myresourcegroup> -ServerName <myserver> -DatabaseName <mydatabase>  
-ReadScale Enabled -Edition Premium
```

### Enabling and disabling Read Scale-Out using the Azure SQL Database REST API

To create a database with read scale-out enabled, or to enable or disable read scale-out for an existing database, create, or update the corresponding database entity 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": "Enabled"
  }
}
```

For more information, see [Databases - Create or Update](#).

## 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 (e.g. 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 will ensure the same load-balancing effect 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 end-point 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

During preview, round-robin or any other load balanced routing between the local replicas of the secondary database is not supported.

## Next steps

- For information about using PowerShell to set read scale-out, see the [Set-AzureRmSqlDatabase](#) or the [New-AzureRmSqlDatabase](#) cmdlets.
- For information about using the REST API to set read scale-out, see [Databases - Create or Update](#).

# Scaling out with Azure SQL Database

9/25/2018 • 6 minutes to read • [Edit Online](#)

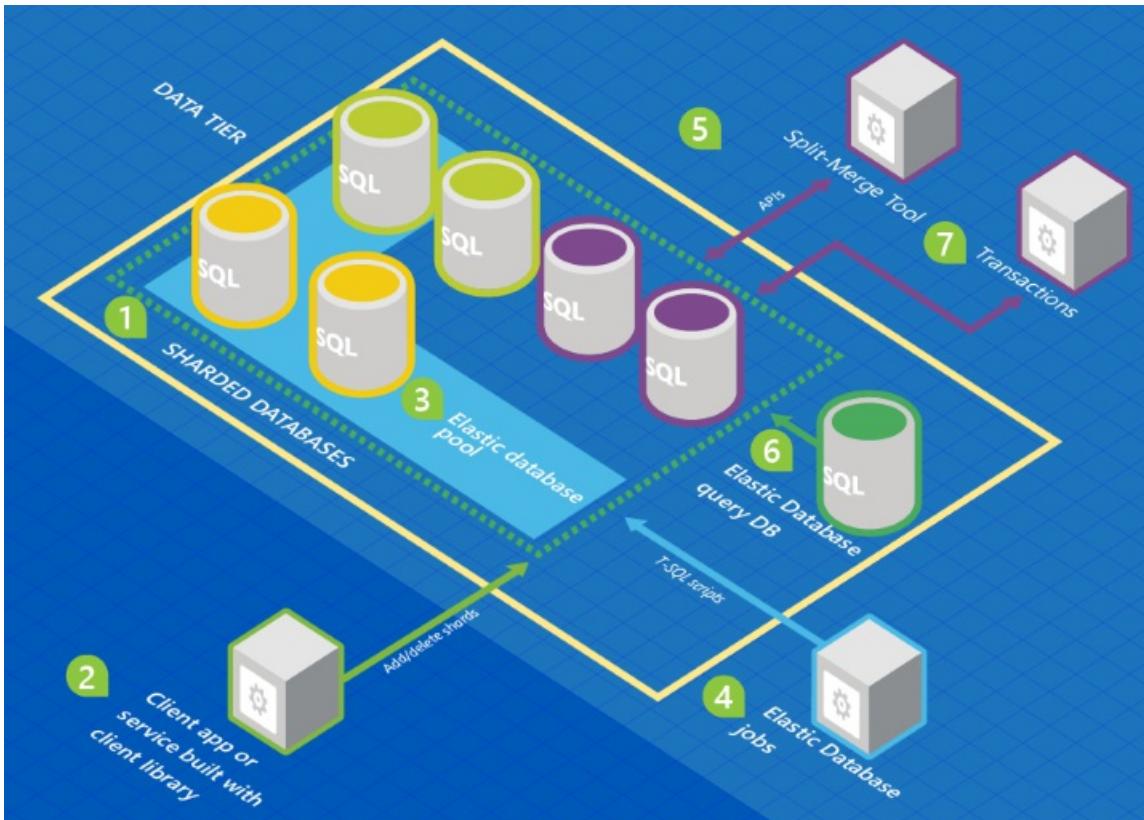
You can easily scale out Azure SQL databases using the **Elastic Database** tools. These tools and features let you use the database resources of **Azure SQL Database** to create solutions for transactional workloads, and especially Software as a Service (SaaS) applications. Elastic Database features 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** (preview): 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?

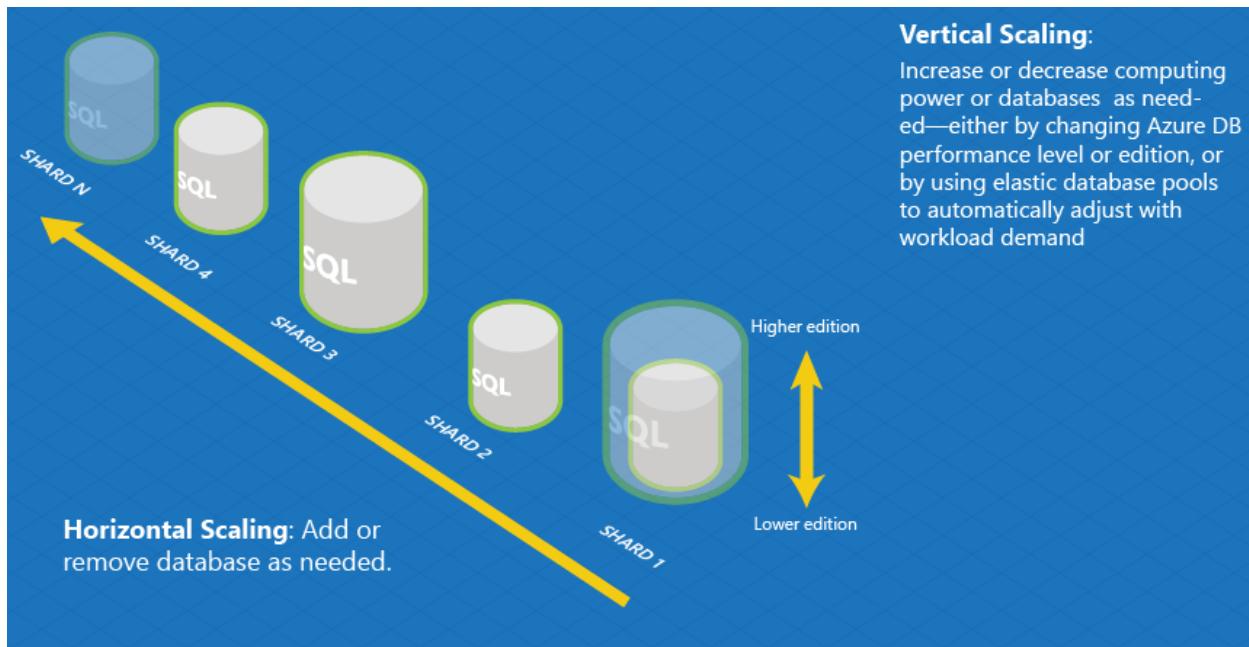
Achieving elasticity and scale for cloud applications has been straightforward for VMs and blob storage - simply add or subtract units, or increase power. But it has remained a challenge for stateful data processing in relational databases. Challenges emerged in these scenarios:

- 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:

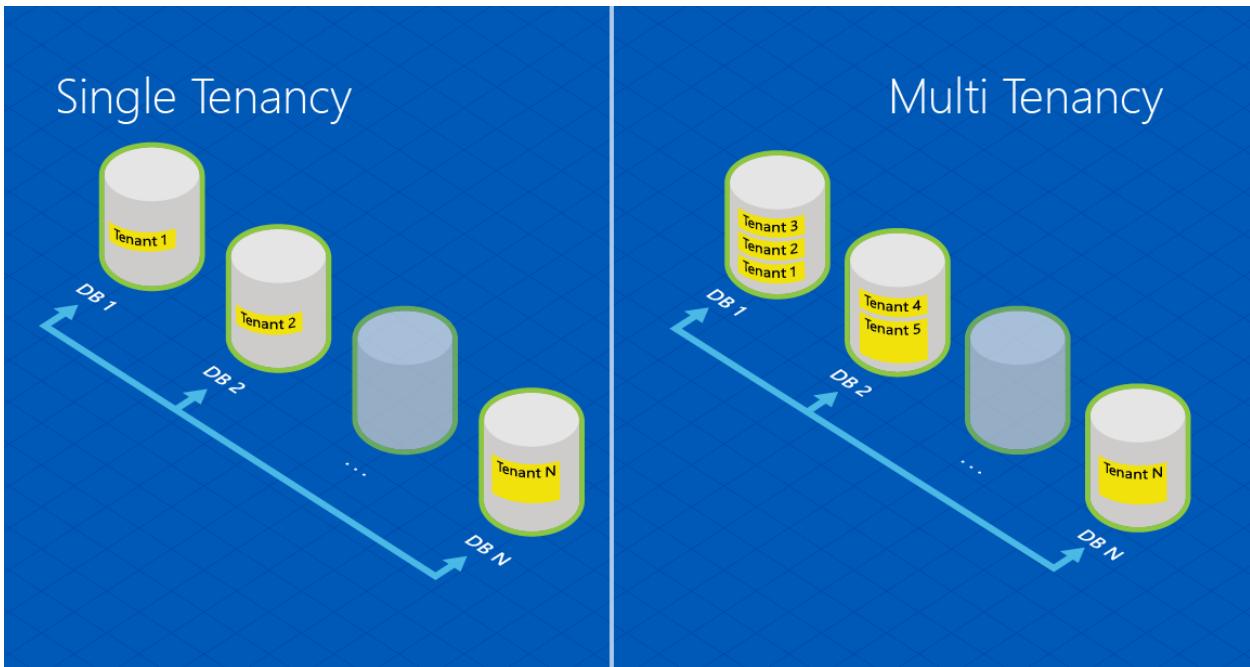
- The total amount of data is too large to fit within the constraints of a single database
- The transaction throughput of the overall workload exceeds the capabilities of a single database
- Tenants may require physical isolation from each other, so separate databases are needed for each tenant
- Different 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](#).

# Building scalable cloud databases

9/25/2018 • 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. [Elastic jobs](#) can then be used to help ease management of these 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](#).

# Upgrade an app to use the latest elastic database client library

9/25/2018 • 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 scriptlet 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](#)

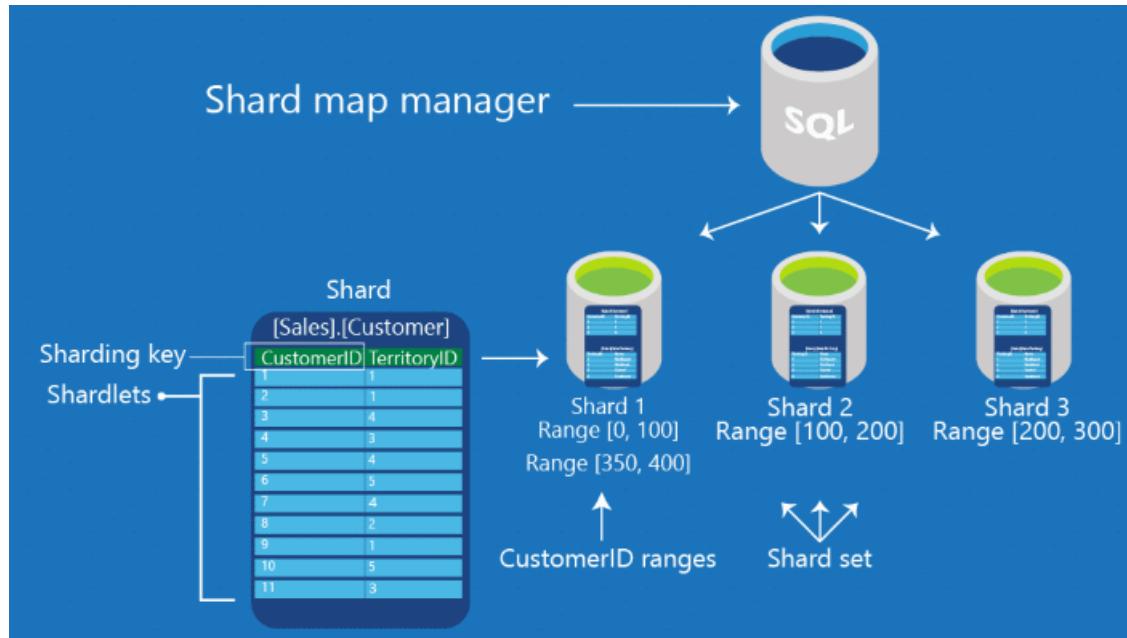
## Additional resources

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](#).

# Scale out databases with the shard map manager

9/25/2018 • 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

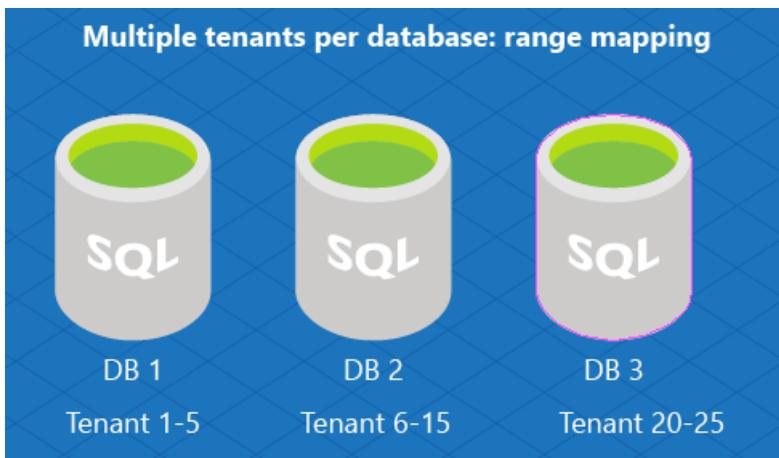
For a single-tenant model, create a **list-mapping** shard map. The single-tenant model assigns one database per tenant. This is an effective model for SaaS developers as it simplifies management.

### Single tenant per database: list mapping



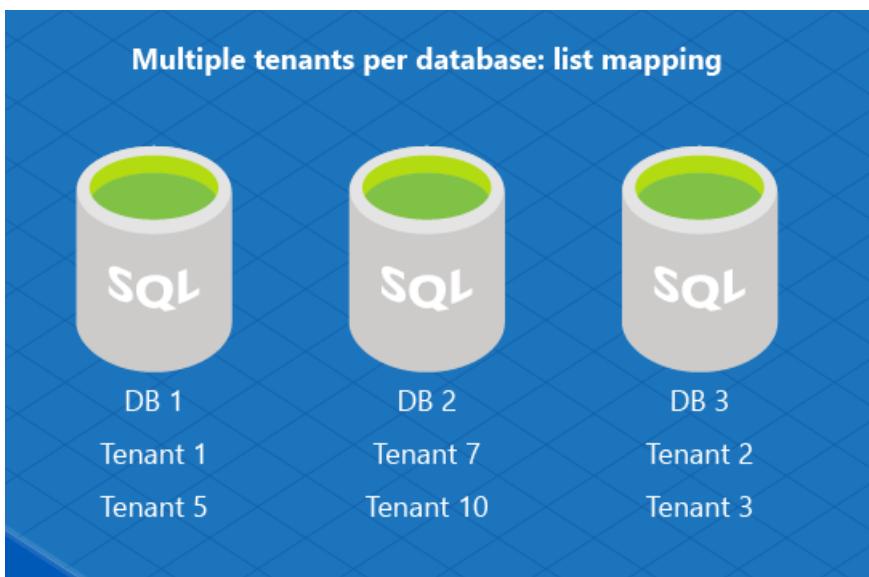
The multi-tenant model assigns several tenants to a single database (and you can distribute groups of tenants across multiple databases). Use this model when you expect each tenant to have small data needs. In this model, assign a range of tenants to a database using **range mapping**.

### Multiple tenants per database: range mapping



Or you can implement a multi-tenant database model using a *list mapping* to assign multiple tenants to a single 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.

### Multiple tenants per database: list mapping



### 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

10/30/2018 • 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 a single 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(smmConnnectionString,
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

9/25/2018 • 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](#)

[Getting started with 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](#).

# Multi-shard querying using elastic database tools

10/8/2018 • 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.GetValue<int>(1);
                var c3Field = sdr.GetValue<Int64>(2);
            }
        }
    }
}
```

A key difference is the construction of multi-shard connections. Where **SqlConnection** operates on a single 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

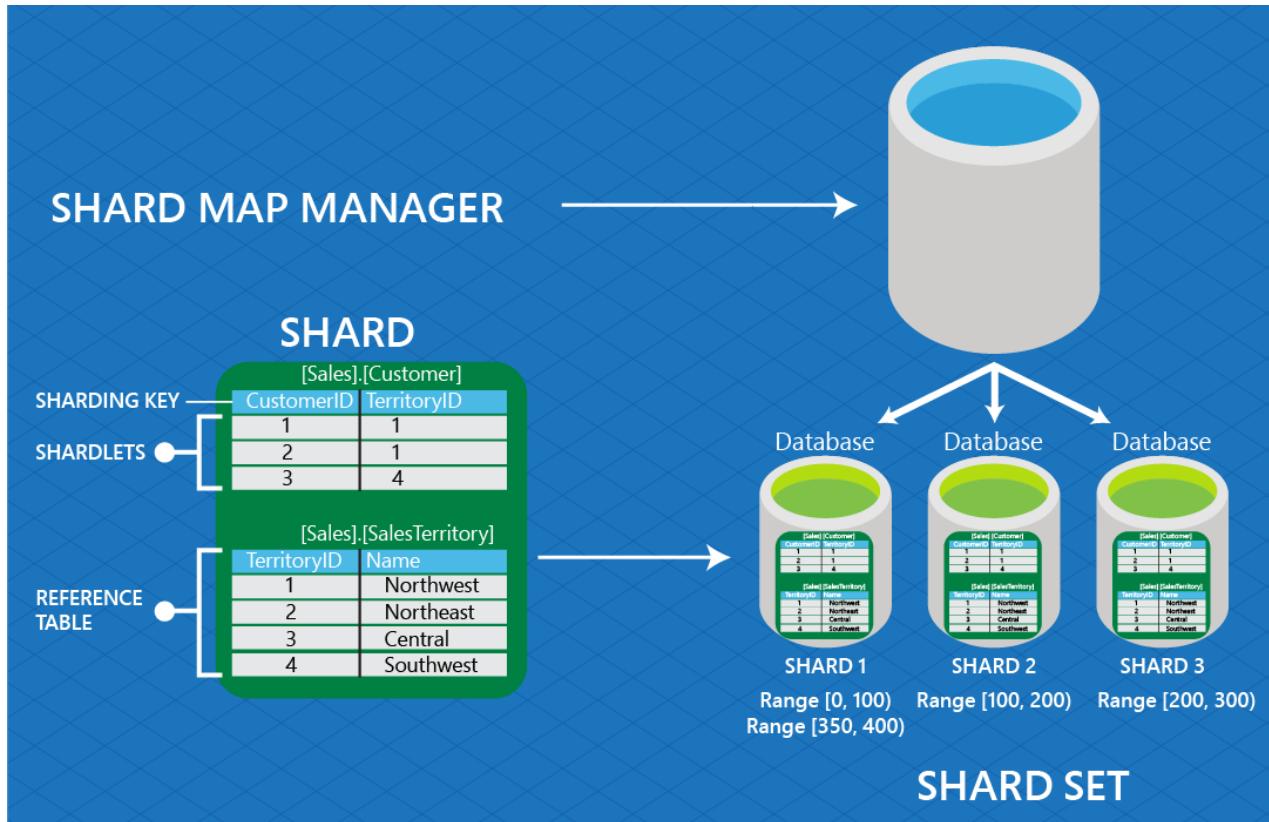
Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Elastic Database tools glossary

10/30/2018 • 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](#).

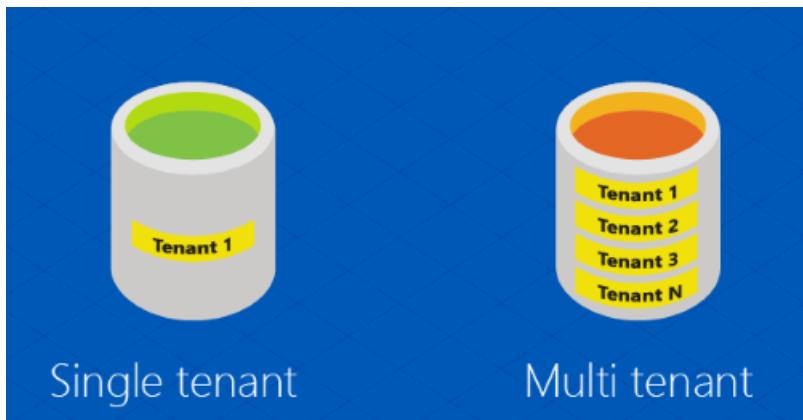
**Global shard map:** The map between sharding keys and their respective shards within a **shard set**. The global shard map is stored in the **shard map manager**. Compare to **local shard map**.

**List shard map:** A shard map in which sharding keys are mapped individually. Compare to [Range Shard Map](#).

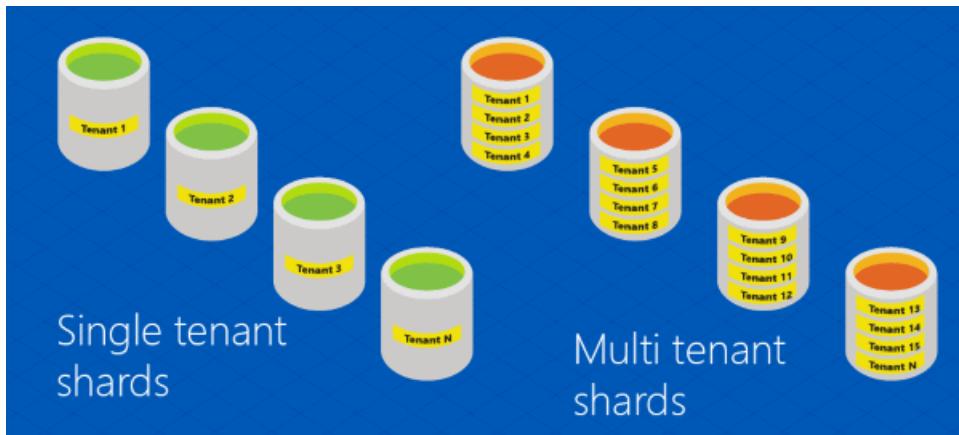
**Local shard map:** Stored on a shard, the local shard map contains mappings for the shardlets that reside on the shard.

**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.

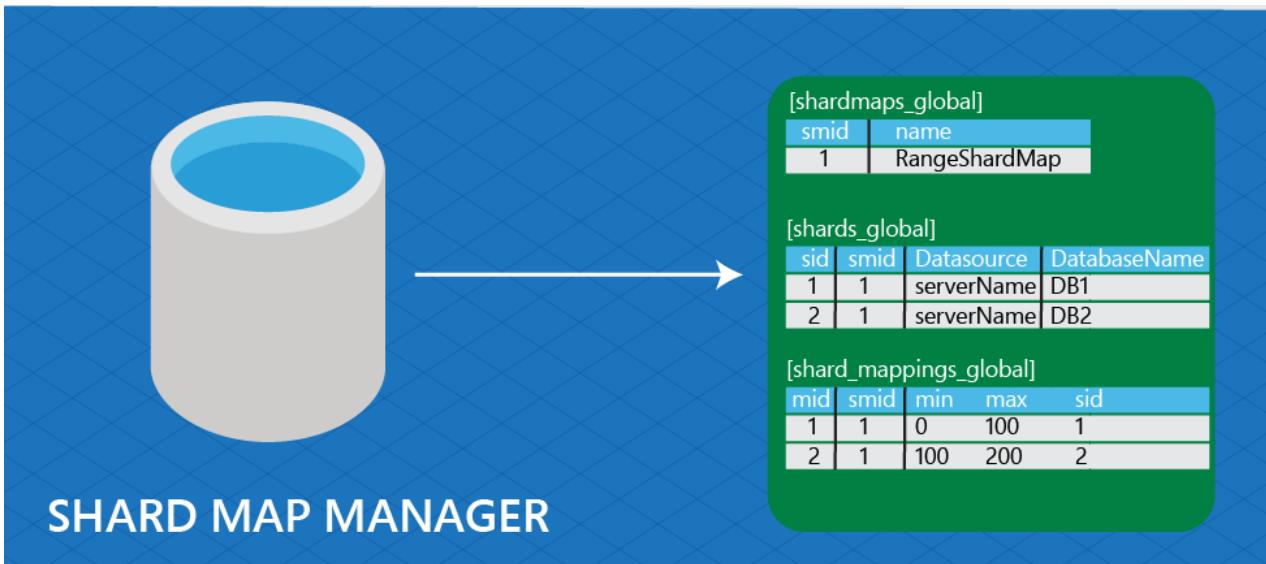
**Sharding key:** A column value that determines how data is distributed across shards. The value type can be one of the following: **int**, **bigint**, **varbinary**, or **uniqueidentifier**.

**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.

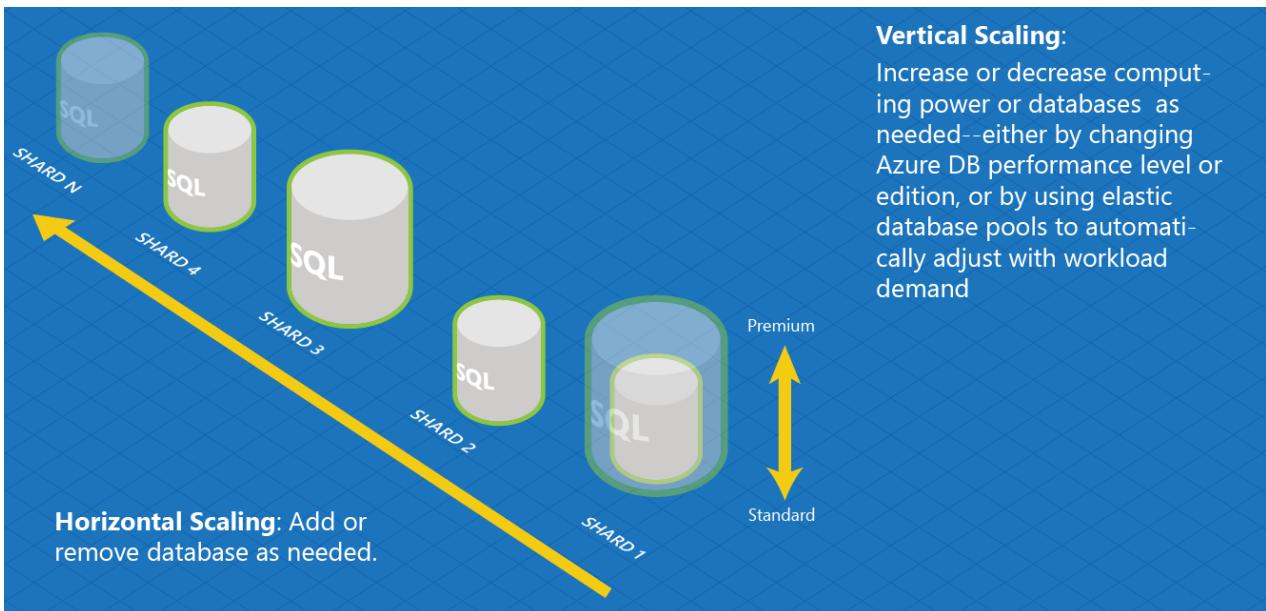
**Shard map:** The set of mappings between sharding keys and their respective shards.

**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

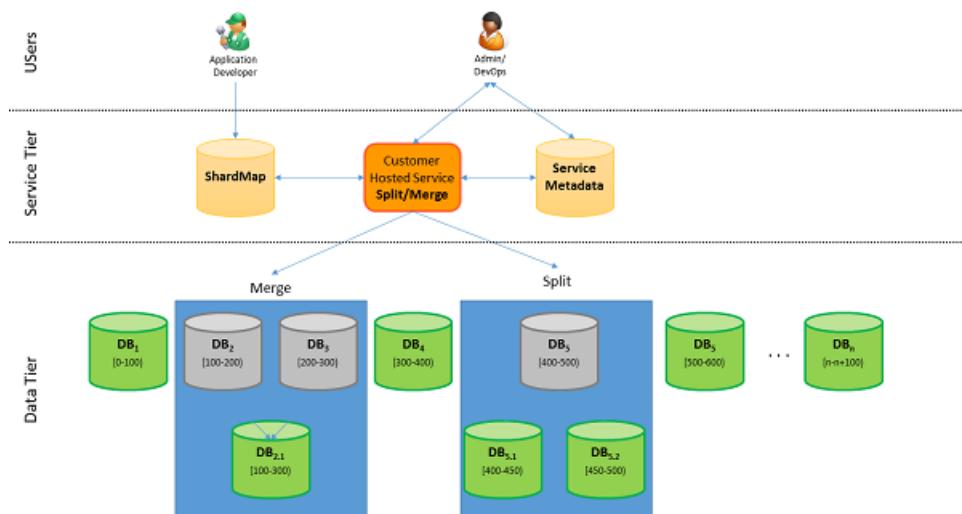
Not using elastic database tools yet? Check out our [Getting Started Guide](#). For questions, please reach out to us on the [SQL Database forum](#) and for feature requests, please add them to the [SQL Database feedback forum](#).

# Moving data between scaled-out cloud databases

10/16/2018 • 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

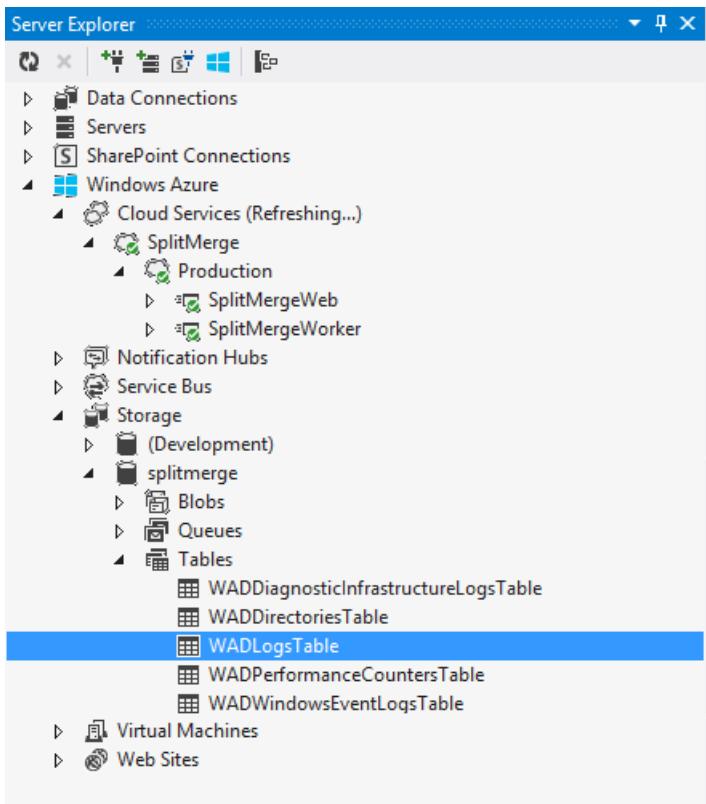
To enable monitoring and diagnostics using the diagnostic configuration for the web and worker roles provided by the NuGet package, run the following commands using Azure PowerShell:

```
$storage_name = "<YourAzureStorageAccount>"  
$key = "<YourAzureStorageAccountKey>"  
$storageContext = New-AzureStorageContext -StorageAccountName $storage_name -StorageAccountKey $key  
$config_path = "<YourFilePath>\SplitMergeWebContent.diagnostics.xml"  
$service_name = "<YourCloudServiceName>"  
Set-AzureServiceDiagnosticsExtension -StorageContext $storageContext -DiagnosticsConfigurationPath  
$config_path -ServiceName $service_name -Slot Production -Role "SplitMergeWeb"  
$config_path = "<YourFilePath>\SplitMergeWorkerContent.diagnostics.xml"  
$service_name = "<YourCloudServiceName>"  
Set-AzureServiceDiagnosticsExtension -StorageContext $storageContext -DiagnosticsConfigurationPath  
$config_path -ServiceName $service_name -Slot Production -Role "SplitMergeWorker"
```

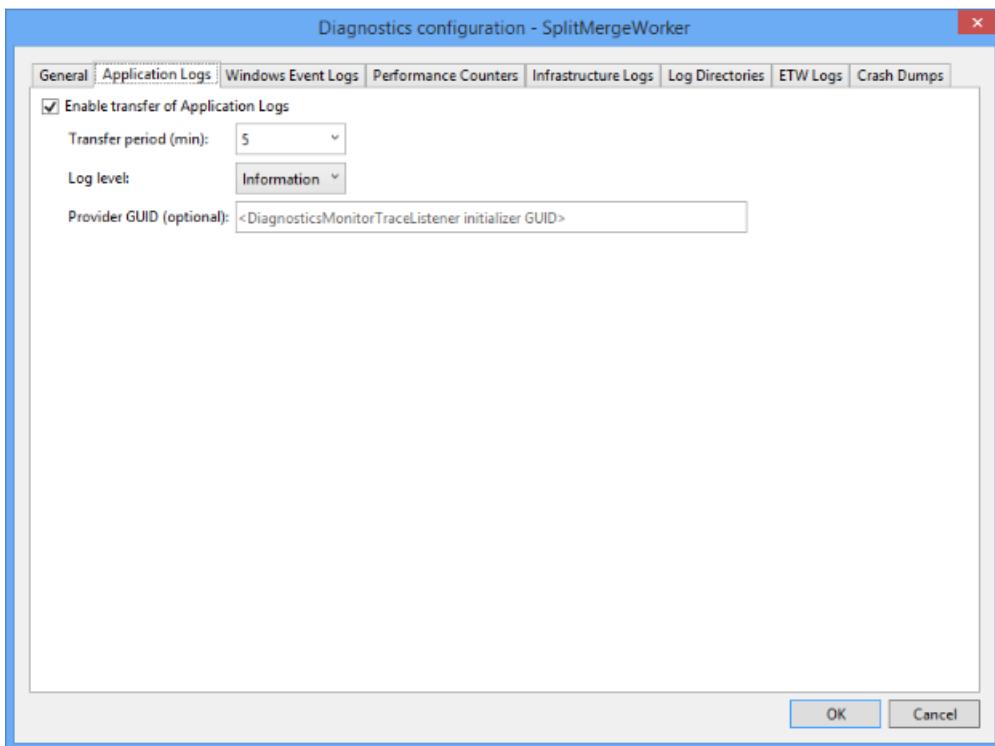
You can find more information on how to configure and deploy diagnostics settings here: [Enabling Diagnostics in Azure Cloud Services and Virtual Machines](#).

## 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](#).

# Using elastic database client library with Dapper

9/25/2018 • 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.

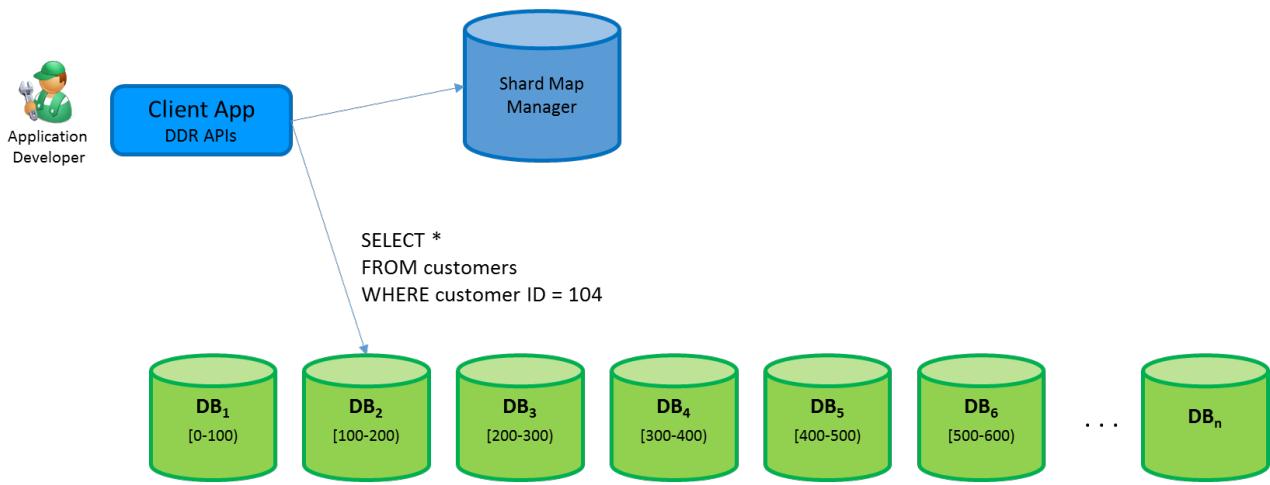
When using DapperExtensions, you no longer need to provide the SQL statements. Extensions methods such as **GetList** or **Insert** over the database connection create the SQL statements behind the scenes.

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 the library 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](#).

# Elastic database tools frequently asked questions (FAQ)

10/30/2018 • 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 [Azure 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 Azure 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](#).

# Get started with Elastic Database Tools

10/30/2018 • 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 logical 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

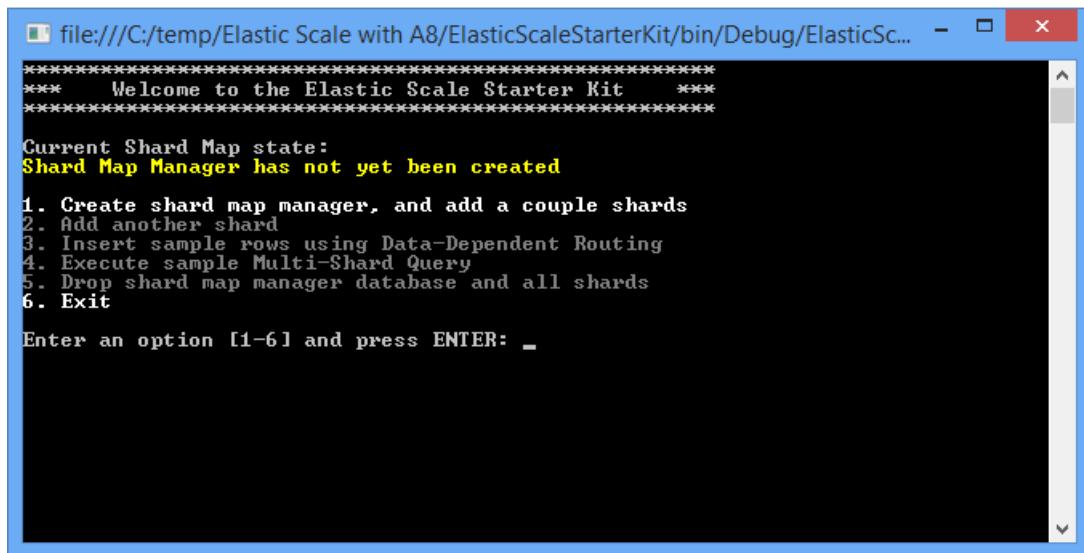
To install the library, go to [Microsoft.Azure.SqlDatabase.ElasticScale.Client](#). The library is installed with the sample app that's described in the following section.

To download and run the sample, follow these steps:

1. Download the [Elastic DB Tools for Azure SQL - Getting Started sample](#) from MSDN. 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.

- 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)

10/30/2018 • 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](#).

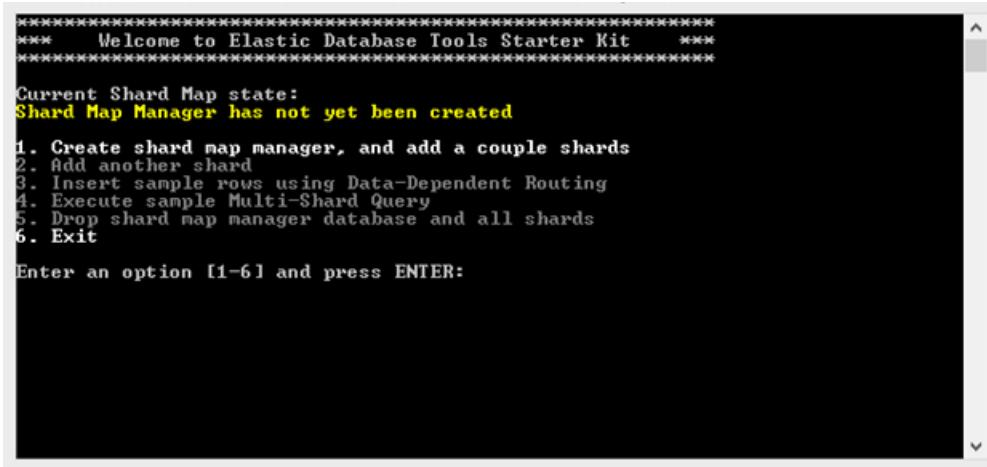
## Prerequisites

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. Follow the steps until step 7 in the section [Download and run the sample app](#). At the end of Step 7, 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

**3 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."

Data + Storage

SQL Database

Choose your pricing tier

Name: ElasticDBQuery

SERVER: [REDACTED] (North Central US)

SELECT SOURCE: Blank database

PRICING TIER: Standard S0

OPTIONAL CONFIGURATION: Collation

RESOURCE GROUP: Group-B

SUBSCRIPTION: ElascticScaleDev\_657854

| P1 Premium   | P2 Premium   | P3 Premium   |
|--|--|--|
| 100 DTUs<br>Up to 500 GB<br>Active Geo-Repli...<br>Point In Time Resto...<br>Auditing  | 200 DTUs<br>Up to 500 GB<br>Active Geo-Repli...<br>Point In Time Resto...<br>Auditing  | 800 DTUs<br>Up to 500 GB<br>Active Geo-Repli...<br>Point In Time Resto...<br>Auditing  |
| 465.00 USD/MONTH (ESTIMATED 31 P1 D...)  | 930.00 USD/MONTH (ESTIMATED 31 P2 D...)  | 3,720.00 USD/MONTH (ESTIMATED 31 P3 D...)  |
| S0 Standard  | S1 Standard  | S2 Standard  |
| 10 DTUs<br>Up to 250 GB<br>Standard Geo-Repli...<br>Point In Time Resto...<br>Auditing | 20 DTUs<br>Up to 250 GB<br>Standard Geo-Repli...<br>Point In Time Resto...<br>Auditing | 50 DTUs<br>Up to 250 GB<br>Standard Geo-Repli...<br>Point In Time Resto...<br>Auditing |
| 15.00 USD/MONTH (ESTIMATED 31 S0 D...)   | 30.00 USD/MONTH (ESTIMATED 31 S1 D...)   | 75.02 USD/MONTH (ESTIMATED 31 S2 D...)   |
| S3 Standard  | B Basic  | W Web (Retired)  |
| 100 DTUs   | 5 DTUs   | - Shared Infrastructure  |

Pin to Startboard

Create Select

#### **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 = '<password>';

CREATE DATABASE SCOPED CREDENTIAL ElasticDBQueryCred
WITH IDENTITY = '<username>',
SECRET = '<password>';
```

"username" and "password" should be the same as login information used in step 6 of [Download and run the sample app](#) in [Getting started with elastic database tools](#).

### **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]
```

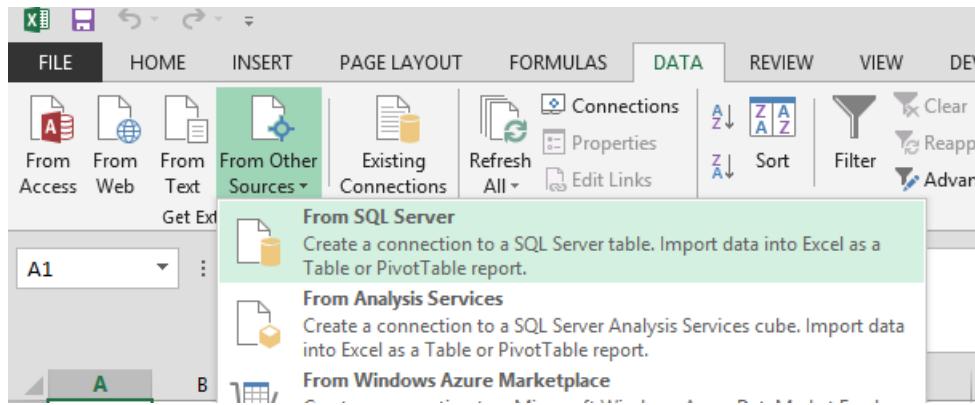
You will notice that the query aggregates results from all the shards and gives the following output:

| Results          | Messages |
|------------------|----------|
| (No column name) |          |
| 1                | 4        |

## Import elastic database query results to Excel

You can import the results from 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.

# Deploy a split-merge service to move data between sharded databases

10/30/2018 • 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=myservername.database.windows.net; Database=mydatabasename;User ID=myuserID;  
Password=mypassword; 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 sha1 -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.

- 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

- Go to the [Azure portal](#)
- Select the cloud service that you created earlier.
- Click **Overview**.
- Choose the staging environment, then click **Upload**.
- 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.
- Ensure that the checkbox labeled **Deploy even if one or more roles contain a single instance** is checked.
- 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=myservername.database.windows.net; Database=mydatabasename;User ID=myuserID;
Password=mypassword; 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. **GetMappings.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 'abcdefg hij.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 'abcdefg hij.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 'abcdefg hij.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/TLS secure 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 **WorkerRoleSynchronizationStorageConnectionString**. 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

9/25/2018 • 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

A 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 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 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" />
```

## 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 (e.g. 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. Please 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 sha1 -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 (e.g. Microsoft Azure Virtual Network).

Change these settings to false in the service configuration file to turn the feature off:

```
<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 sha1 -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 his/hers exclusive use and should choose his/hers 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 sha1 -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 he/she 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 his/hers 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 which 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

10/30/2018 • 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](#).

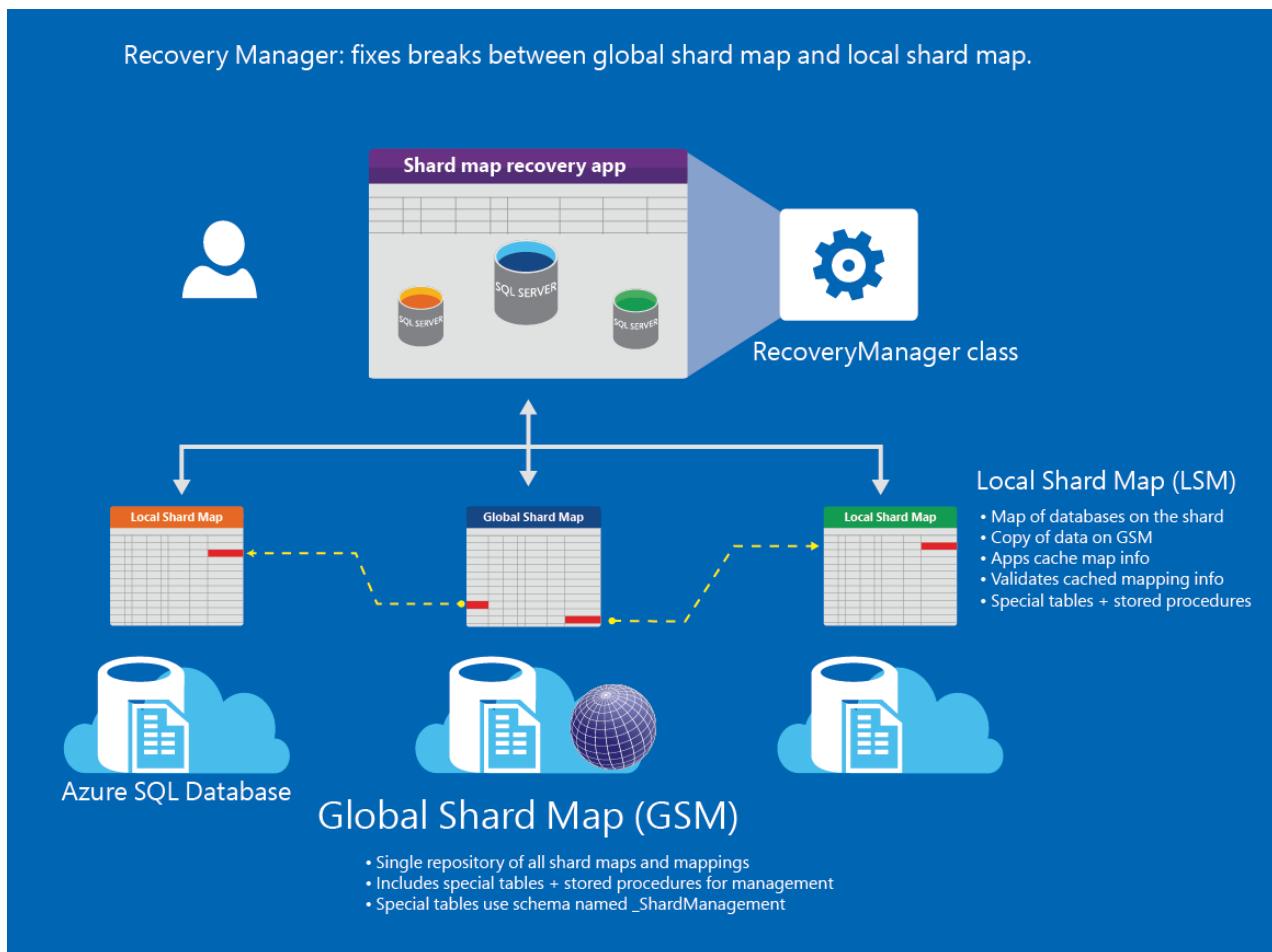
# Using the RecoveryManager class to fix shard map problems

10/30/2018 • 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

9/25/2018 • 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>'  
-SqlDatabaseName '<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:

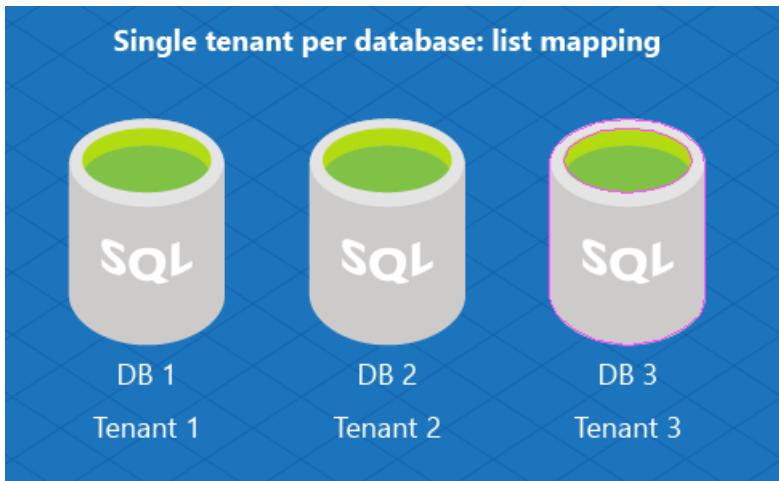
1. Single tenant per database (For terms, see the [glossary](#).)

2. Multiple tenants per database (two types):

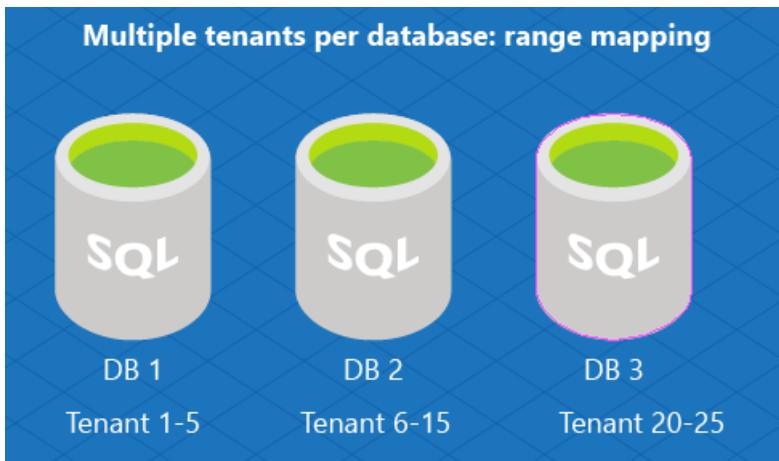
a. List mapping

b. Range mapping

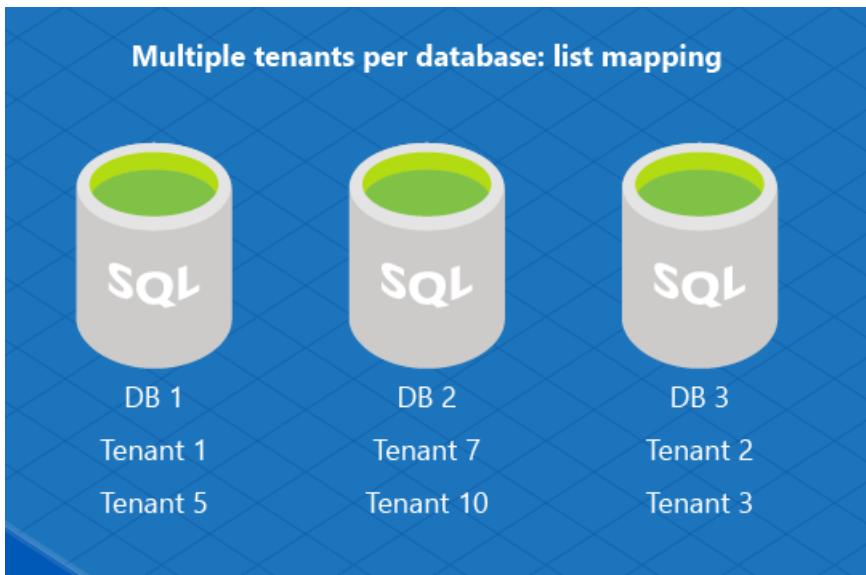
For a single-tenant model, create a **list mapping** shard map. The single-tenant model assigns one database per tenant. This is an effective model for SaaS developers as it simplifies management.



The multi-tenant model assigns several tenants to a single 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 a single 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 a single 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 a single 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](#).

# Performance counters for shard map manager

9/25/2018 • 2 minutes to read • [Edit Online](#)

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.

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.

**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

9/25/2018 • 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](#) from MSDN. 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 connectionStr)
        : base(CreateDDRConnection(shardMap, shardingKey, connectionStr),
              true /* contextOwnsConnection */)
    {
    }

    // Only static methods are allowed in calls into base class c'tors.
    private static DbConnection CreateDDRConnection(
        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 = shardMap.OpenConnectionForKey<T>
            (shardingKey, connectionStr, 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 intialization 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](#).

# Feature comparison: Azure SQL Database versus SQL Server

10/8/2018 • 6 minutes to read • [Edit Online](#)

Azure SQL Database shares a common code base with SQL Server. The features of SQL Server supported by Azure SQL Database depend on the type of Azure SQL database that you create. With Azure SQL Database, you can either create a database as part of a [managed instance](#) or you can create a database that is part of Logical server and optionally placed in an Elastic pool.

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.

## SQL Server feature support in Azure SQL Database

The following table lists the major features of SQL Server and provides information about whether the feature is partially or fully supported and a link to more information about the feature.

| SQL FEATURE                     | SUPPORTED IN AZURE SQL DATABASE/LOGICAL SERVER   | SUPPORTED IN AZURE SQL DATABASE/MANAGED INSTANCE (BUSINESS CRITICAL TIER IS IN PREVIEW)  |
|---------------------------------|--|--|
| Always Encrypted                | Yes - see <a href="#">Cert store</a> and <a href="#">Key vault</a>   | Yes - see <a href="#">Cert store</a> and <a href="#">Key vault</a>   |
| Always On Availability Groups   | High availability is included with every database. Disaster recovery is discussed in <a href="#">Overview of business continuity with Azure SQL Database</a> | High availability is included with every database. Disaster recovery is discussed in <a href="#">Overview of business continuity with Azure SQL Database</a> |
| Attach a database               | No   | No   |
| Application roles               | Yes  | Yes  |
| Auditing                        | Yes  | Yes  |
| Automatic backups               | Yes  | Yes  |
| Automatic tuning (plan forcing) | Yes  | Yes  |
| Automatic tuning (indexes)      | Yes  | No   |
| Azure Data Studio               | Yes  | Yes  |
| BACPAC file (export)            | Yes - see <a href="#">SQL Database export</a>  | No   |
| BACPAC file (import)            | Yes - see <a href="#">SQL Database import</a>  | No   |

| SQL FEATURE                                       | SUPPORTED IN AZURE SQL DATABASE/LOGICAL SERVER  | SUPPORTED IN AZURE SQL DATABASE/MANAGED INSTANCE<br>(BUSINESS CRITICAL TIER IS IN PREVIEW)                       |
|---|---|--|
| <a href="#">BACKUP command</a>                    | No, only system-initiated automatic backups - see <a href="#">Automated backups</a>                 | System-initiated automated backups and user initiated copy-only backups - see <a href="#">Backup differences</a> |
| <a href="#">Built-in functions</a>                | Most - see individual functions   | Yes - see <a href="#">Stored procedures, functions, triggers differences</a>                                     |
| <a href="#">Change data capture</a>               | No  | Yes  |
| <a href="#">Change tracking</a>                   | Yes   | Yes  |
| <a href="#">Collation statements</a>              | Yes   | Yes  |
| <a href="#">Columnstore indexes</a>               | Yes - Premium tier, Standard tier - S3 and above, General Purpose tier, and Business Critical tiers | Yes  |
| <a href="#">Common language runtime (CLR)</a>     | No  | Yes - see <a href="#">CLR differences</a>  |
| <a href="#">Contained databases</a>               | Yes   | Yes  |
| <a href="#">Contained users</a>                   | Yes   | Yes  |
| <a href="#">Control of flow language keywords</a> | Yes   | Yes  |
| <a href="#">Cross-database queries</a>            | No - see <a href="#">Elastic queries</a>  | Yes, plus <a href="#">Elastic queries</a>  |
| <a href="#">Cross-database transactions</a>       | No  | Yes - see <a href="#">Linked server differences</a>  |
| <a href="#">Cursors</a>                           | Yes   | Yes  |
| <a href="#">Data compression</a>                  | Yes   | Yes  |
| <a href="#">Database mail</a>                     | No  | Yes  |
| <a href="#">Data Migration Service (DMS)</a>      | Yes   | Yes  |
| <a href="#">Database mirroring</a>                | No  | No   |
| <a href="#">Database configuration settings</a>   | Yes   | Yes  |
| <a href="#">Data Quality Services (DQS)</a>       | No  | No   |
| <a href="#">Database snapshots</a>                | No  | No   |
| <a href="#">Data types</a>                        | Yes   | Yes  |
| <a href="#">DBCC statements</a>                   | Most - see individual statements  | Yes - see <a href="#">DBCC differences</a>   |
| <a href="#">DDL statements</a>                    | Most - see individual statements  | Yes - see <a href="#">T-SQL differences</a>  |

| SQL FEATURE                       | SUPPORTED IN AZURE SQL DATABASE/LOGICAL SERVER                 | SUPPORTED IN AZURE SQL DATABASE/MANAGED INSTANCE<br>(BUSINESS CRITICAL TIER IS IN PREVIEW)  |
|-----------------------------------|--|---|
| DDL triggers                      | Database only  | Yes   |
| Distributed partition views       | No   | Yes   |
| Distributed transactions - MS DTC | No - see <a href="#">Elastic transactions</a>                  | No - see <a href="#">Elastic transactions</a>   |
| DML statements                    | Yes  | Yes   |
| DML triggers                      | Most - see individual statements                               | Yes   |
| DMVs                              | Most - see individual DMVs                                     | Yes - see <a href="#">T-SQL differences</a>   |
| Dynamic data masking              | Yes  | Yes   |
| Elastic pools                     | Yes  | Built-in - a single Managed Instance can have multiple databases that share the same pool of resources  |
| Event notifications               | No - see <a href="#">Alerts</a>                                | Yes   |
| Expressions                       | Yes  | Yes   |
| Extended events                   | Some - see <a href="#">Extended events in SQL Database</a>     | Yes - see <a href="#">Extended events differences</a>   |
| Extended stored procedures        | No   | No  |
| Files and file groups             | Primary file group only  | Yes   |
| Filestream                        | No   | No  |
| Full-text search                  | Third-party word breakers are not supported                    | Third-party word breakers are not supported   |
| Functions                         | Most - see individual functions                                | Yes - see <a href="#">Stored procedures, functions, triggers differences</a>  |
| Geo-restore                       | Yes - General Purpose and Business Critical service tiers only | No – you can restore COPY_ONLY full backups that you take periodically - see <a href="#">Backup differences</a> and <a href="#">Restore differences</a> . |
| Geo-replication                   | Yes - General Purpose and Business Critical service tiers only | No  |
| Graph processing                  | Yes  | Yes   |
| In-memory optimization            | Yes - Premium and Business Critical tiers only                 | Yes - Business Critical tier only - currently in preview  |
| JSON data support                 | Yes  | Yes   |

| SQL FEATURE                    | SUPPORTED IN AZURE SQL DATABASE/LOGICAL SERVER   | SUPPORTED IN AZURE SQL DATABASE/MANAGED INSTANCE<br>(BUSINESS CRITICAL TIER IS IN PREVIEW)   |
|--------------------------------|--|--|
| Language elements              | Most - see individual elements   | Yes - see <a href="#">T-SQL differences</a>  |
| Linked servers                 | No - see <a href="#">Elastic query</a>   | Only to SQL Server and SQL Database  |
| Log shipping                   | <a href="#">High availability</a> is included with every database. Disaster recovery is discussed in <a href="#">Overview of business continuity with Azure SQL Database</a> | <a href="#">High availability</a> is included with every database. Disaster recovery is discussed in <a href="#">Overview of business continuity with Azure SQL Database</a> |
| Master Data Services (MDS)     | No   | No   |
| Minimal logging in bulk import | No   | No   |
| Modifying system data          | No   | Yes  |
| Online index operations        | Yes  | Yes  |
| OPENDATASOURCE                 | No   | Yes - see <a href="#">T-SQL differences</a>  |
| OPENJSON                       | Yes  | Yes  |
| OPENQUERY                      | No   | Yes - see <a href="#">T-SQL differences</a>  |
| OPENROWSET                     | No   | Yes - see <a href="#">T-SQL differences</a>  |
| OPENXML                        | Yes  | Yes  |
| Operators                      | Most - see individual operators  | Yes - see <a href="#">T-SQL differences</a>  |
| Partitioning                   | Yes  | Yes  |
| Point in time database restore | Yes - General Purpose and Business Critical service tiers only - see <a href="#">SQL Database recovery</a>   | Yes - see <a href="#">SQL Database recovery</a>  |
| Polybase                       | No   | No   |
| Policy-based management        | No   | No   |
| Predicates                     | Yes  | Yes  |
| R Services                     | Preview release; see <a href="#">What's new in machine learning</a>  | No   |
| Resource governor              | No   | Yes  |
| RESTORE statements             | No   | Yes - see <a href="#">Restore differences</a>  |

| SQL FEATURE                            | SUPPORTED IN AZURE SQL DATABASE/LOGICAL SERVER  | SUPPORTED IN AZURE SQL DATABASE/MANAGED INSTANCE<br>(BUSINESS CRITICAL TIER IS IN PREVIEW)  |
|--|---|---|
| Restore database from backup           | From automated backups only - see <a href="#">SQL Database recovery</a>   | From automated backups - see <a href="#">SQL Database recovery</a> and from full backups - see <a href="#">Backup differences</a>   |
| Row Level Security                     | Yes   | Yes   |
| Semantic search                        | No  | No  |
| Sequence numbers                       | Yes   | Yes   |
| Service Broker                         | No  | Yes - see <a href="#">Service Broker differences</a>  |
| Server configuration settings          | No  | Yes - see <a href="#">T-SQL differences</a>   |
| Set statements                         | Most - see individual statements  | Yes - see <a href="#">T-SQL differences</a>   |
| SMO                                    | Yes   | Yes   |
| Spatial                                | Yes   | Yes   |
| SQL Data Sync                          | Yes   | No  |
| SQL Server Agent                       | No - see <a href="#">Elastic jobs</a>   | Yes - see <a href="#">SQL Server Agent differences</a>  |
| SQL Server Analysis Services (SSAS)    | No - see <a href="#">Azure Analysis Services</a>  | No - see <a href="#">Azure Analysis Services</a>  |
| SQL Server Auditing                    | No - see <a href="#">SQL Database auditing</a>  | Yes - see <a href="#">Auditing differences</a>  |
| SQL Server Data Tools (SSDT)           | Yes   | Yes   |
| SQL Server Integration Services (SSIS) | 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 <a href="#">Create Azure-SSIS IR in ADF</a> . | 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 <a href="#">Create Azure-SSIS IR in ADF</a> . |
|  | To compare the SSIS features in SQL Database logical server and Managed Instance, see <a href="#">Compare SQL Database logical server and Managed Instance</a> .  | To compare the SSIS features in SQL Database and Managed Instance, see <a href="#">Compare SQL Database logical server and Managed Instance</a> .   |
| SQL Server Management Studio (SSMS)    | Yes   | Yes   |
| SQL Server PowerShell                  | Yes   | Yes   |
| SQL Server Profiler                    | No - see <a href="#">Extended events</a>  | Yes   |
| SQL Server Replication                 | Transactional and snapshot replication subscriber only  | Yes - <a href="#">Replication with SQL Database Managed Instance - public preview</a>   |

| SQL FEATURE                          | SUPPORTED IN AZURE SQL DATABASE/LOGICAL SERVER   | SUPPORTED IN AZURE SQL DATABASE/MANAGED INSTANCE<br>(BUSINESS CRITICAL TIER IS IN PREVIEW)   |
|--------------------------------------|--|--|
| SQL Server Reporting Services (SSRS) | No - see <a href="#">Power BI</a>  | No - see <a href="#">Power BI</a>  |
| Stored procedures                    | Yes  | Yes  |
| System stored functions              | Most - see individual functions  | Yes - see <a href="#">Stored procedures, functions, triggers differences</a>   |
| System stored procedures             | Some - see individual stored procedures  | Yes - see <a href="#">Stored procedures, functions, triggers differences</a>   |
| System tables                        | Some - see individual tables   | Yes - see <a href="#">T-SQL differences</a>  |
| System catalog views                 | Some - see individual views  | Yes - see <a href="#">T-SQL differences</a>  |
| Temporary tables                     | Local and database-scoped global temporary tables  | Local and instance-scoped global temporary tables  |
| Temporal tables                      | Yes  | Yes  |
| Threat detection                     | Yes  | Yes  |
| Trace flags                          | No   | No   |
| Variables                            | Yes  | Yes  |
| Transparent data encryption (TDE)    | Yes - General Purpose and Business Critical service tiers only   | Partial, only with service-managed encryption  |
| VNet                                 | Partial - see <a href="#">VNet Endpoints</a>   | Yes, Resource Manager model only   |
| Windows Server Failover Clustering   | <a href="#">High availability</a> is included with every database. Disaster recovery is discussed in <a href="#">Overview of business continuity with Azure SQL Database</a> | <a href="#">High availability</a> is included with every database. Disaster recovery is discussed in <a href="#">Overview of business continuity with Azure SQL Database</a> |
| XML indexes                          | Yes  | Yes  |

## Next steps

- For information about the Azure SQL Database service, see [What is SQL Database?](#)
- For information about a Managed Instance, see [What is a Managed Instance?](#)

# Monitoring and performance tuning

10/23/2018 • 7 minutes to read • [Edit Online](#)

Azure SQL Database is automatically managed and flexible data service where you can easily monitor usage, add or remove resources (CPU, memory, I/O), find recommendations that can improve performance of your database, or let database adapt to your workload and automatically optimize performance.

## Monitoring database performance

Monitoring the performance of a SQL database in Azure starts with monitoring the resource utilization relative to the level of database performance you choose. Azure SQL Database enables you to 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 these tuning recommendations to improve performance of your workload. You can also let Azure SQL database to [automatically optimize performance of your queries](#) by applying all identified recommendations and verifying that they improve database performance.

You have the following options for monitoring and troubleshooting database performance:

- In the [Azure portal](#), click **SQL databases**, select the database, and then use the Monitoring chart to look for resources approaching their maximum. DTU consumption is shown by default. Click **Edit** to change the time range and values shown.
- Use [Query Performance Insight](#) to identify the queries that spend the most of resources.
- Use [SQL Database Advisor](#) to view recommendations for creating and dropping indexes, parameterizing queries, and fixing schema issues.
- Use [Azure SQL Intelligent Insights](#) for automatic monitoring of your database performance. Once a performance issue is detected, a diagnostic log is generated with details and Root Cause Analysis (RCA) of the issue. Performance improvement recommendation is provided when possible.
- [Enable automatic tuning](#) and let Azure SQL database automatically fix identified performance issues.
- Use [dynamic management views \(DMVs\)](#), [extended events](#), and the [Query Store](#) for more detailed troubleshooting of performance issues.

### TIP

See [performance guidance](#) to find techniques that you can use to improve performance of Azure SQL Database after identifying the performance issue using one or more of the above methods.

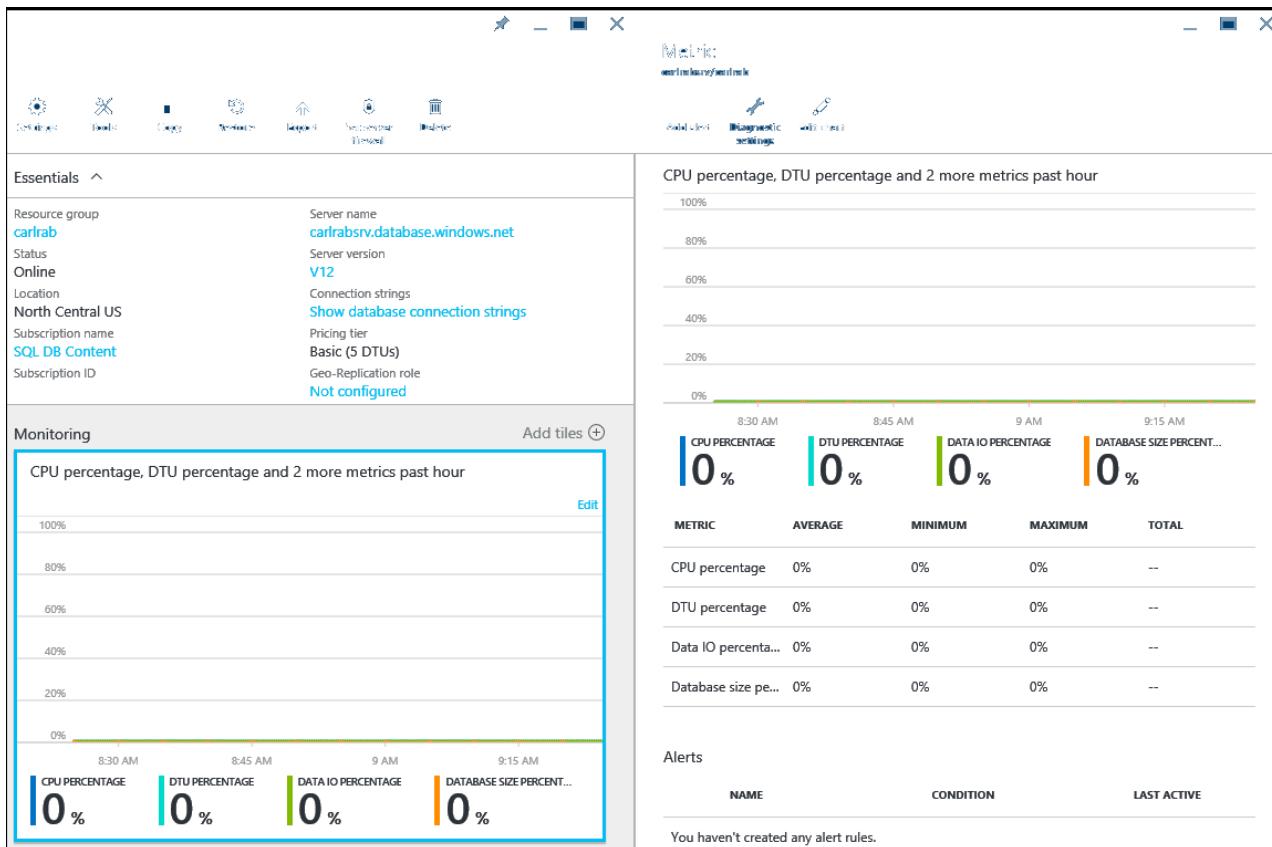
## Monitor databases using the Azure portal

In the [Azure portal](#), you can monitor a single databases 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.

## Troubleshoot performance issues

To diagnose and resolve performance issues, begin by understanding the state of each active query and the conditions that cause performance issues relevant to each workload state. To improve Azure SQL Database performance, understand that each active query request from your application is either in a running or a waiting state. When troubleshooting a performance issue in Azure SQL Database, keep the following chart in mind as you read through this article to diagnose and resolve performance issues.

# Azure SQL Database

## Workload State

Running

Waiting

Execution

Compilation

Blocking

I/O

Tempdb

Memory grants

For a workload with performance issues, the performance issue may be due to CPU contention (a **running-related** condition) or individual queries are waiting on something (a **waiting-related** condition).

### Running-related performance issues

As a general guideline, if your CPU utilization is consistently at or above 80%, you have a running-related performance issue. If you have a running-related issue, it may be caused by insufficient CPU resources or it may be related to one of the following conditions:

- Too many running queries
- Too many compiling queries
- One or more executing queries are using a sub-optimal query plan

If you determine that you have a running-related performance issue, your goal is to identify the precise issue using one or more methods. The most common methods for identifying running-related issues are:

- Use the [Azure portal](#) to monitor CPU percentage utilization.
- Use the following [dynamic management views](#):
  - `sys.dm_db_resource_stats` returns CPU, I/O, 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. Historical data is maintained for one hour.
  - `sys.resource_stats` returns CPU usage and storage data for an Azure SQL Database. The data is collected and aggregated within five-minute intervals.

#### IMPORTANT

For a set a T-SQL queries using these DMVs to troubleshoot CPU utilization issues, see [Identify CPU performance issues](#).

Once you identify the issue, you can either tune the problem queries or upgrade the compute size or service tier to increase the capacity of your Azure SQL database to absorb the CPU requirements. For information on scaling resources for single databases, see [Scale single database resources in Azure SQL Database](#) and for scaling resources for elastic pools, see [Scale elastic pool resources in Azure SQL Database](#). For information on scaling a

managed instance, see [Instance-level resource limits](#).

## Waiting-related performance issues

Once you are certain that you are not facing a high-CPU, running-related performance issue, you are facing a waiting-related performance issue. Namely, your CPU resources are not being used efficiently because the CPU is waiting on some other resource. In this case, your next step is to identify what your CPU resources are waiting on. The most common methods for showing the top wait type categories:

- The [Query Store](#) provides wait statistics per query over time. In Query Store, wait types are combined into wait categories. The mapping of wait categories to wait types is available in [sys.query\\_store\\_wait\\_stats](#).
- [sys.dm\\_db\\_wait\\_stats](#) returns information about all the waits encountered by threads that executed during operation. You can use this aggregated view to diagnose performance issues with Azure SQL Database and also with specific queries and batches.
- [sys.dm\\_os\\_waiting\\_tasks](#) returns information about the wait queue of tasks that are waiting on some resource.

As shown in the previous chart, the most common waits are:

- Locks (blocking)
- I/O
- `tempdb`-related contention
- Memory grant waits

### IMPORTANT

For a set a T-SQL queries using these DMVs to troubleshoot these waiting-related issues, see:

- [Identify I/O performance issues](#)
- [Identify `tempdb` performance issues](#)
- [Identify memory grant waits](#)

## Improving database performance with more resources

Finally, if there are no actionable items that can improve performance of your database, you can change the amount of resources available in Azure SQL Database. You can 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](#), you can change either the service tier or increase the resources allocated to your database.

1. For single databases, you can [change service tiers](#) or [compute resources](#) on-demand to improve database performance.
2. For multiple databases, consider using [elastic pools](#) to scale resources automatically.

## Tune and refactor application or database code

You can change application code to more optimally use the database, change indexes, force plans, or use hints to manually adapt the database to your workload. Find some guidance and tips for manual tuning and rewriting the code in the [performance guidance topic](#) article.

## Next steps

- To enable automatic tuning in Azure SQL Database and let automatic tuning feature fully manage your workload, see [Enable automatic tuning](#).

- To use manual tuning, you can review [Tuning recommendations in Azure portal](#) and manually apply the ones that improve performance of your queries.
- Change resources that are available in your database by changing [Azure SQL Database service tiers](#)

# Monitor and improve performance

9/25/2018 • 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.

To review your database performance, use the **Performance** tile on the Overview page, or navigate down to "Support + troubleshooting" section:

The screenshot shows the Microsoft Azure Workload Insights Demo dashboard for a database named 'WorkloadInsightsDemo'. The left sidebar contains navigation links for various database features like Quick start, Pricing tier, Geo-Replication, and Performance overview. A red box highlights the 'Performance overview' link under the 'SUPPORT + TROUBLESHOOTING' section. The main content area is divided into three sections: 'Monitoring' (DTU percentage today chart), 'Operations' (Performance recommendations summary), and 'Geo-Replication' (world map). The 'Operations' section is also highlighted with a red box. It displays 5 active recommendations with counts for High (2), Medium (0), and Low (3) priority levels. A 'Security' section is visible next to it, showing an 'Enable Threat Detection' button.

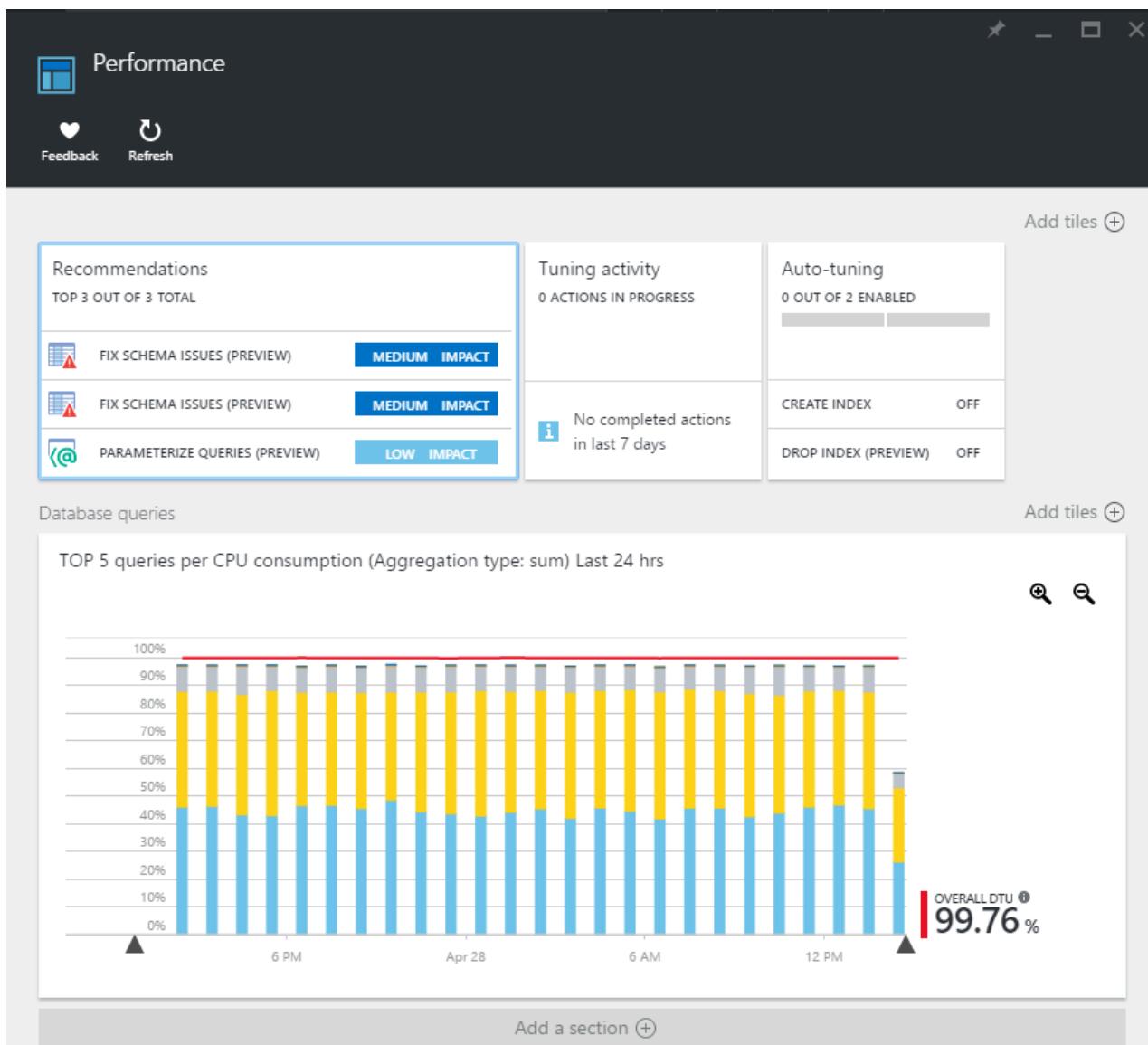
In the "Support + troubleshooting" section, you can use the following pages:

1. [Performance overview](#) to monitor performance of your database.
2. [Performance recommendations](#) to find performance recommendations that can improve performance of your workload.
3. [Query Performance Insight](#) to find top resource consuming queries.
4. [Automatic tuning](#) to let Azure SQL Database automatically optimize your database.

## 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.

The screenshot shows the Microsoft Azure Workload Insights Demo - Performance recommendations page. On the left, there's a sidebar with various navigation links like Overview, Activity log, Tags, and Diagnose and solve problems. The main content area has sections for Recommendations and Tuning history.

**Recommendations**

| Action               | Recommendation Description   | Impact      |
|----------------------|--|-------------|
| CREATE INDEX         | Table: [Employees]<br>Indexed columns: [City], [State]                                     | HIGH IMPACT |
| PARAMETERIZE QUERIES | Scope: Entire database<br>Reason: Non-parameterized queries are causing performance issues | HIGH IMPACT |
| CREATE INDEX         | Table: [DataPoints]<br>Indexed columns: [Name],[Money],[Power]                             | LOW IMPACT  |
| DROP INDEX           | Index name: IX_FF<br>Reason: Unused index  | LOW IMPACT  |
| DROP INDEX           | Index name: MyIndex123<br>Reason: Duplicate index  | LOW IMPACT  |

**Tuning history**

| Action       | Recommendation Description   | Status  | Time                 |
|--------------|--|---------|----------------------|
| CREATE INDEX | Initiated by: User<br>Table: [DataPoints]<br>Indexed columns: [Name],[Money] | Success | 5/19/2017 2:42:12 PM |
| DROP INDEX   | Initiated by: User<br>Index name: MyIndex321<br>Reason: Duplicate index      | Success | 5/19/2017 2:42:12 PM |
| DROP INDEX   | Initiated by: System<br>Index name: IX_FF<br>Reason: Duplicate index         | Success | 5/18/2017 2:42:13 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.

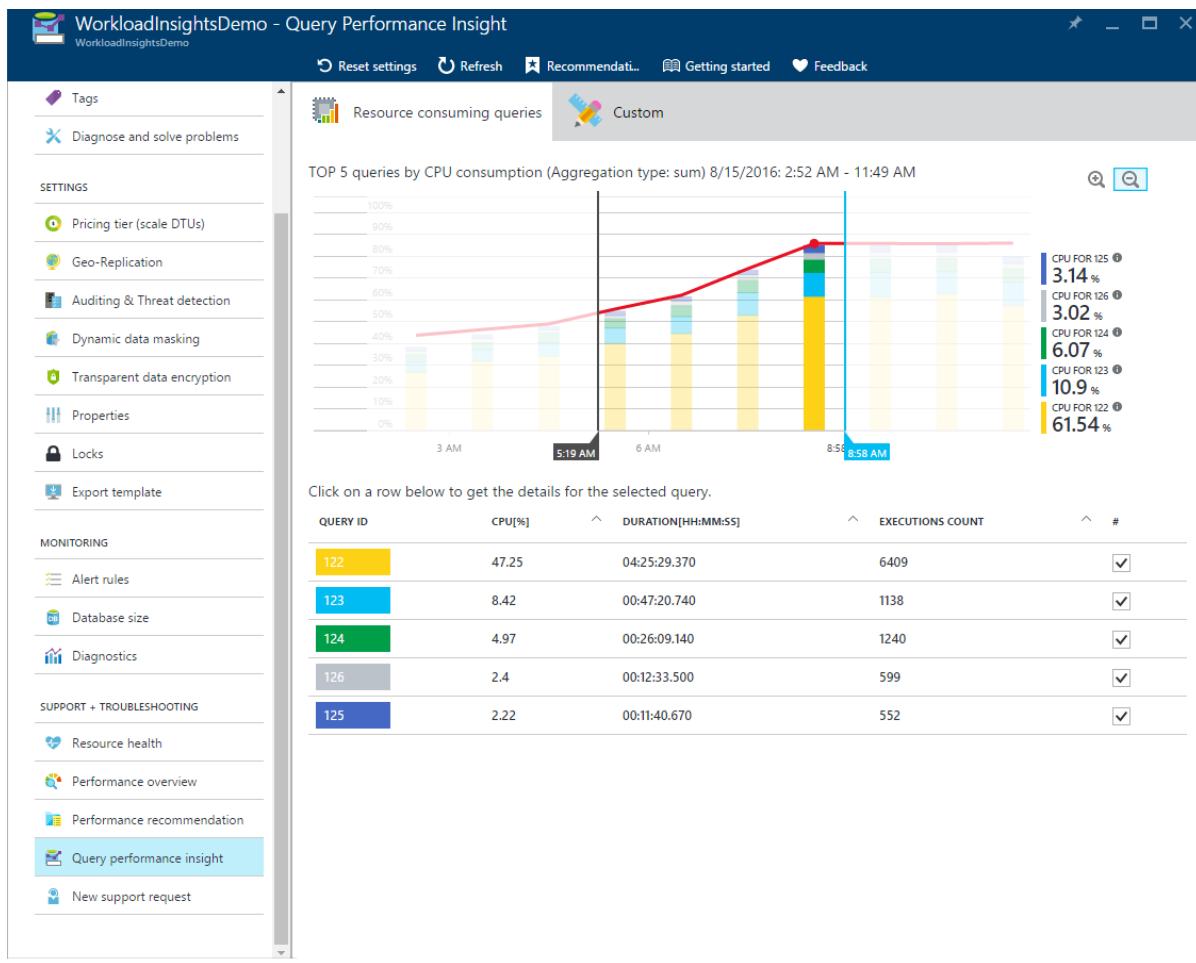
## Automatic tuning

Azure SQL databases can automatically tune database performance by applying [performance recommendations](#). To learn more, read [Automatic tuning article](#). To enable it, read [how to enable automatic tuning](#).

## 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](#).

## Additional resources

- [Azure SQL Database performance guidance for single databases](#)
- [When should an elastic pool be used?](#)

# Azure SQL Database Query Performance Insight

9/25/2018 • 7 minutes to read • [Edit Online](#)

Managing and tuning the performance of relational databases is a challenging task that requires significant expertise and time investment. Query Performance Insight allows you to spend less time troubleshooting database performance by providing the following:

- Deeper insight into your databases resource (DTU) consumption.
- The top queries by CPU/Duration/Execution count, which can potentially be tuned for improved performance.
- The ability to drill down into the details of a query, view its text and history of resource utilization.
- Performance tuning annotations that show actions performed by [SQL Azure Database Advisor](#)

## Prerequisites

- Query Performance Insight requires that [Query Store](#) is active on your database. If Query Store is not running, the portal prompts you to turn it on.

## Permissions

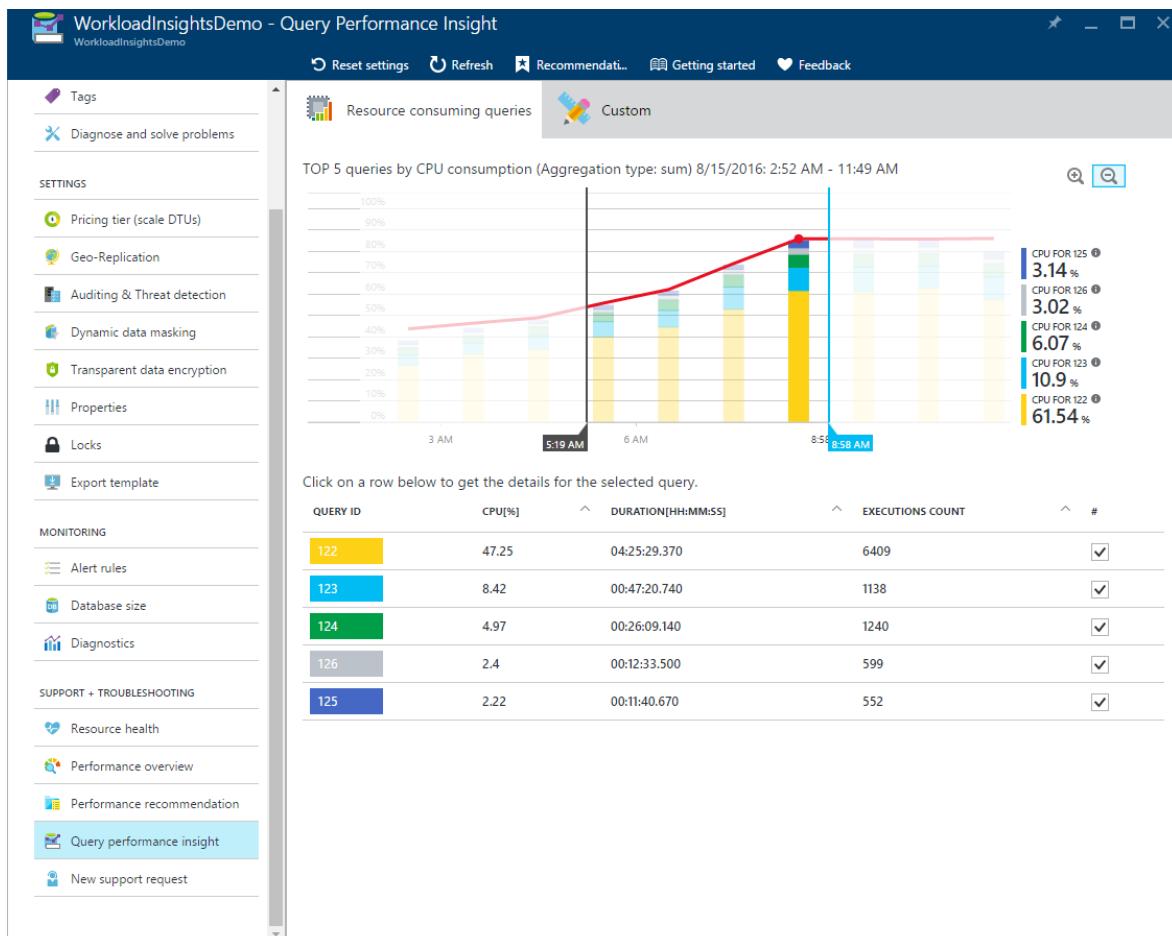
The following [role-based access control](#) permissions are required to use Query Performance Insight:

- **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.

## Using Query Performance Insight

Query Performance Insight is easy to use:

- Open [Azure portal](#) and find database that you want to examine.
  - From left-hand side menu, under support and troubleshooting, select "Query Performance Insight".
- On the first tab, review the list of top resource-consuming queries.
- Select an individual query to view its details.
- Open [SQL Azure Database Advisor](#) and check if any recommendations are available.
- Use sliders or zoom icons to change observed interval.



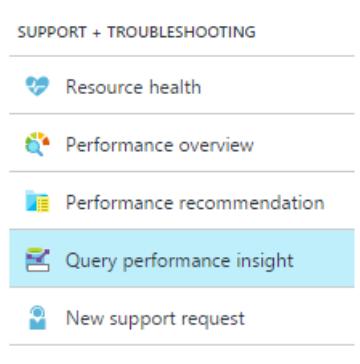
#### NOTE

A couple hours of data needs to be captured by Query Store for SQL Database to provide query performance insights. If the database has no activity or Query Store was not active during a certain time period, the charts will be empty when displaying that time period. You may enable Query Store at any time if it is not running.

## Review top CPU consuming queries

In the [portal](#) do the following:

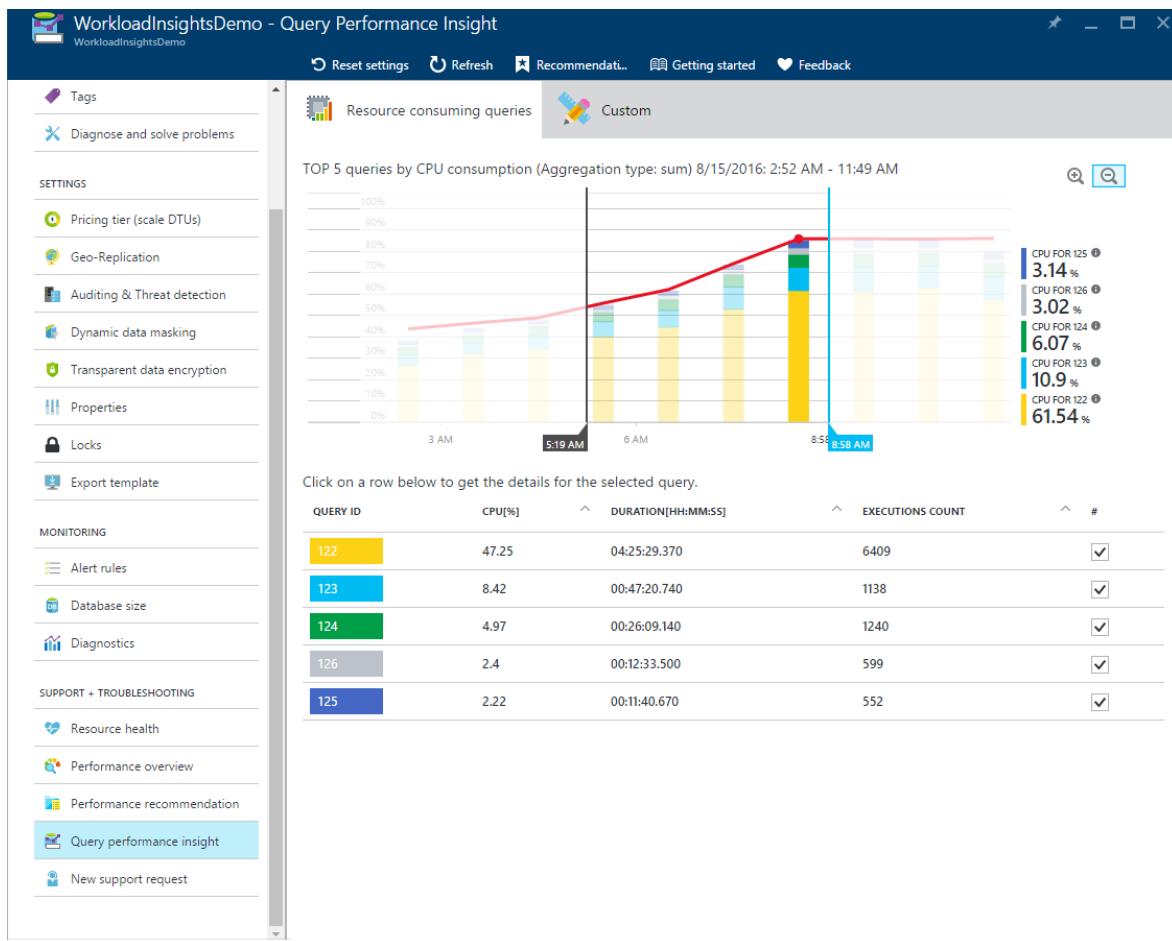
1. Browse to a SQL database and click **All settings > Support + Troubleshooting > Query performance insight**.



The top queries view opens and the top CPU consuming queries are listed.

2. Click around the chart for details.

The top line shows overall DTU% for the database, while the bars show CPU% consumed by the selected queries during the selected interval (for example, if **Past week** is selected each bar represents one day).

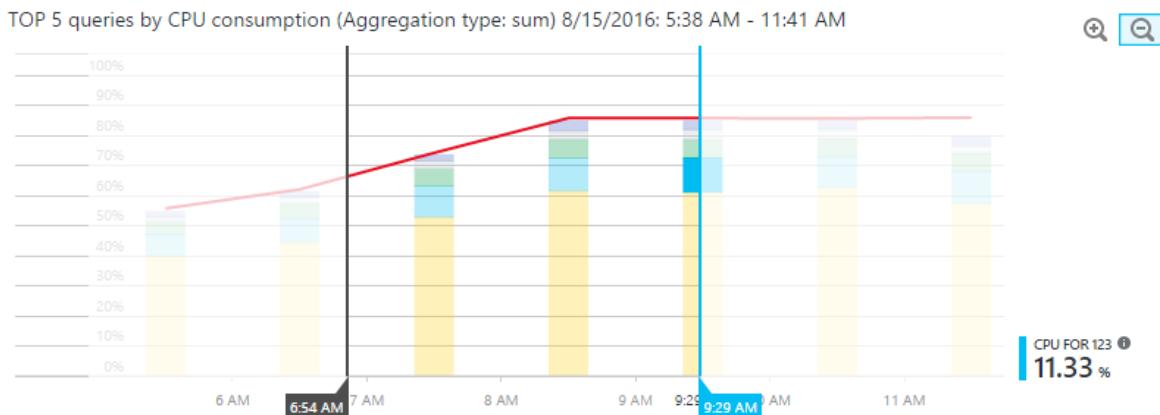


The bottom grid represents aggregated information for the visible queries.

- Query ID - unique identifier of query inside database.
- CPU per query during observable interval (depends on aggregation function).
- Duration per query (depends on aggregation function).
- Total number of executions for a particular query.

Select or clear individual queries to include or exclude them from the chart using checkboxes.

3. If your data becomes stale, click the **Refresh** button.
4. You can use sliders and zoom buttons to change observation interval and investigate spikes:



5. Optionally, if you want a different view, you can select **Custom** tab and set:

- Metric (CPU, duration, execution count)
- Time interval (Last 24 hours, Past week, Past month).
- Number of queries.
- Aggregation function.

The screenshot shows the 'Resource consuming queries' blade. At the top left is a bar chart icon and the text 'Resource consuming queries'. To its right is a 'Custom' icon. Below these are four filter sections: 'Metric type' set to 'CPU', 'Time period' set to 'Past week', 'Number of queries' set to '20', and 'Aggregation type' set to 'max'. A blue 'Go >' button is located at the bottom right of the filter area.

## Viewing individual query details

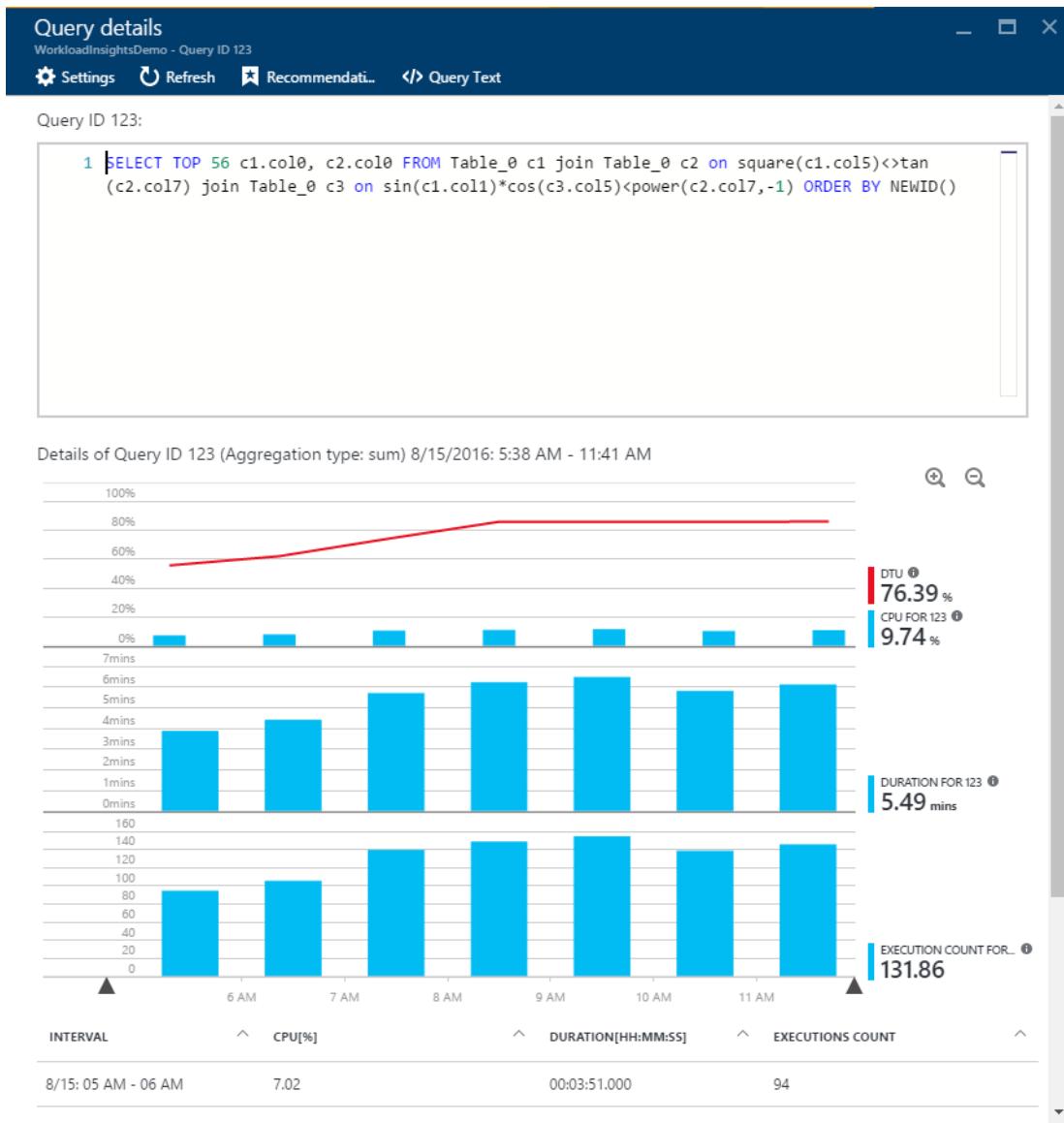
To view query details:

1. Click 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"/> |

2. The details view opens and the queries CPU consumption/Duration/Execution count is broken down over time.
3. Click around the chart for details.
  - Top chart shows line with overall database DTU%, and the bars are CPU% consumed by the selected query.
  - Second chart shows total duration by the selected query.
  - Bottom chart shows total number of executions by the selected query.



4. Optionally, use sliders, zoom buttons or click **Settings** to customize how query data is displayed, or to pick a different time period.

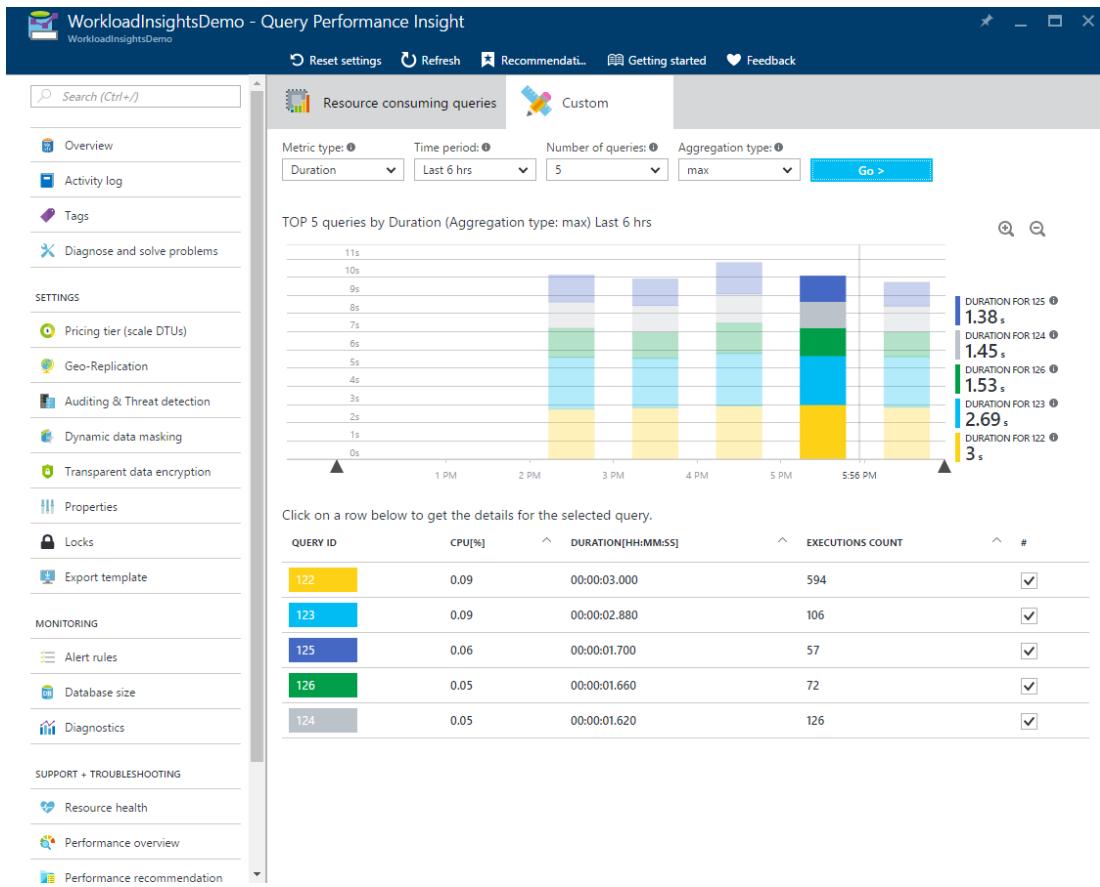
## Review top queries per duration

In the recent update of Query Performance Insight, we introduced two new metrics that can help you identify potential bottlenecks: duration and execution count.

Long-running queries have the greatest potential for locking resources longer, blocking other users, and limiting scalability. They are also the best candidates for optimization.

To identify long running queries:

1. Open **Custom** tab in Query Performance Insight for selected database
2. Change metrics to be **duration**
3. Select number of queries and observation interval
4. Select aggregation function
  - **Sum** adds up all query execution time during whole observation interval.
  - **Max** finds queries which execution time was maximum at whole observation interval.
  - **Avg** finds average execution time of all query executions and show you the top out of these averages.



## Review top queries per execution count

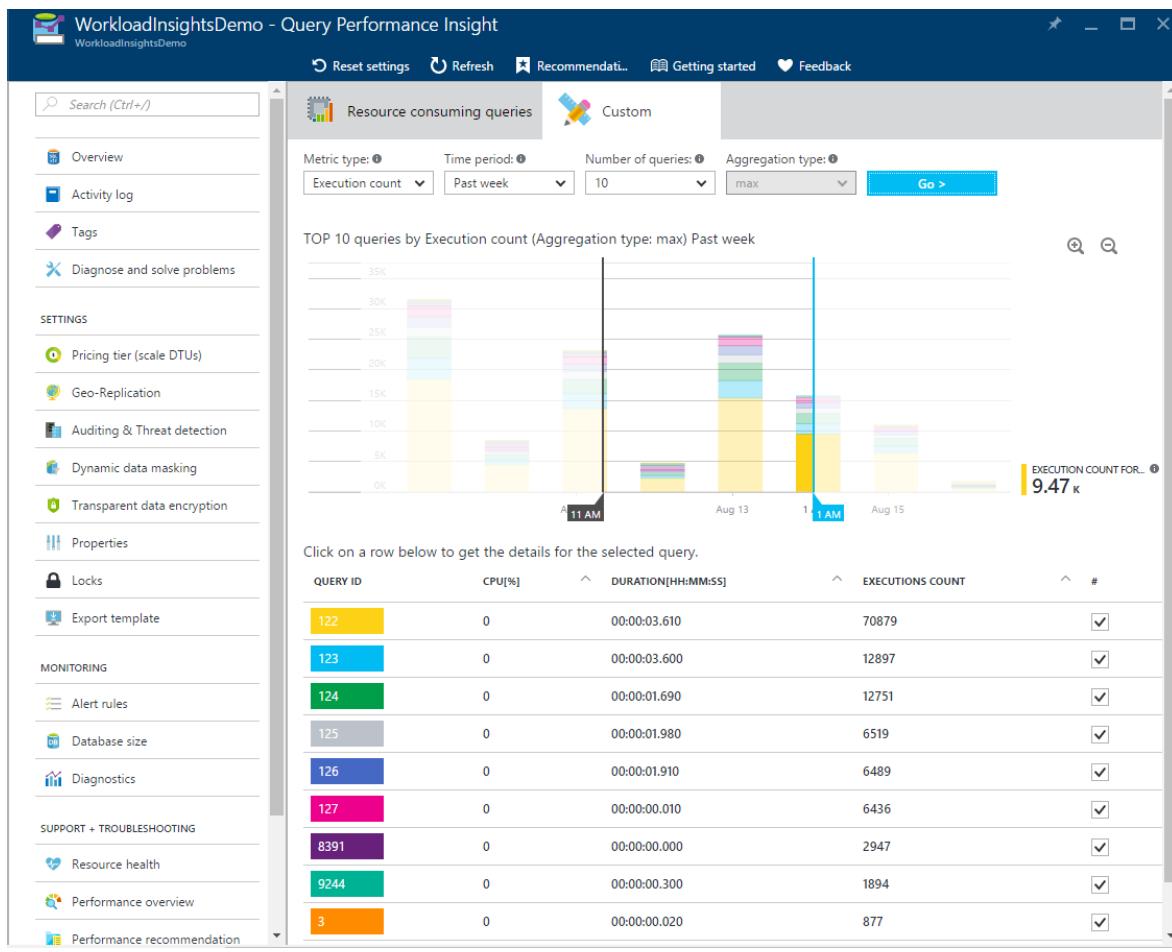
High number of executions might not be affecting database itself and resources usage can be low, but overall application might get slow.

In some cases, very high execution count may lead to increase of network round trips. Round trips significantly affect performance. They are subject to network latency and to downstream server latency.

For example, many data-driven Web sites heavily access the database for every user request. While connection pooling helps, the increased network traffic and processing load on the database server can adversely affect performance. General advice is to keep round trips to an absolute minimum.

To identify frequently executed queries ("chatty") queries:

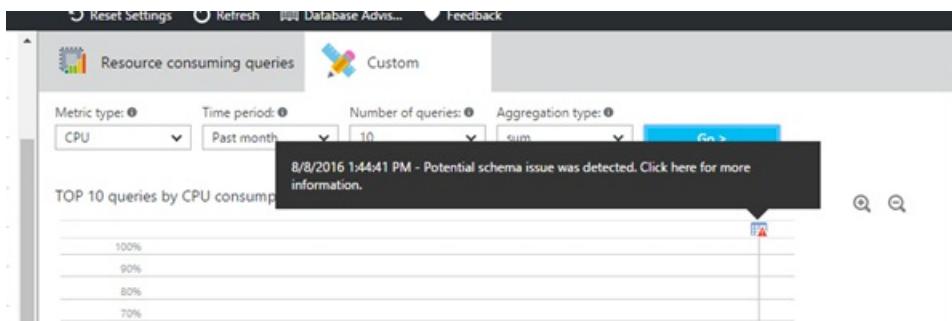
1. Open **Custom** tab in Query Performance Insight for selected database
2. Change metrics to be **execution count**
3. Select number of queries and observation interval



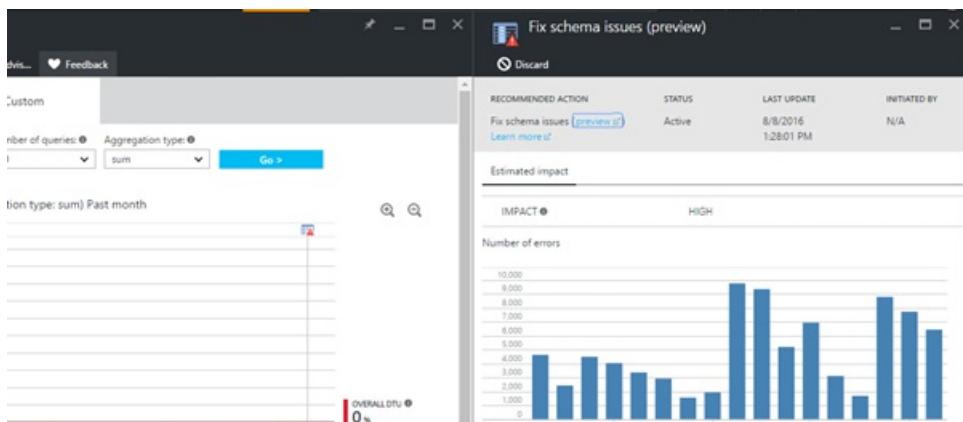
## Understanding performance tuning annotations

While exploring your workload in Query Performance Insight, you might notice icons with vertical line on top of the chart.

These icons are annotations; they represent performance affecting actions performed by [SQL Azure Database Advisor](#). By hovering annotation, you get basic information about the action:



If you want to know more or apply advisor recommendation, click the icon. It will open details of action. If it's an active recommendation you can apply it straight away using command.



### Multiple annotations.

It's possible, that because of zoom level, annotations that are close to each other will get collapsed into one. This will be represented by special icon, clicking it will open new blade where list of grouped annotations will be shown. Correlating queries and performance tuning actions can help to better understand your workload.

## Optimizing the Query Store configuration for Query Performance Insight

During your use of Query Performance Insight, you might encounter the following Query Store 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."

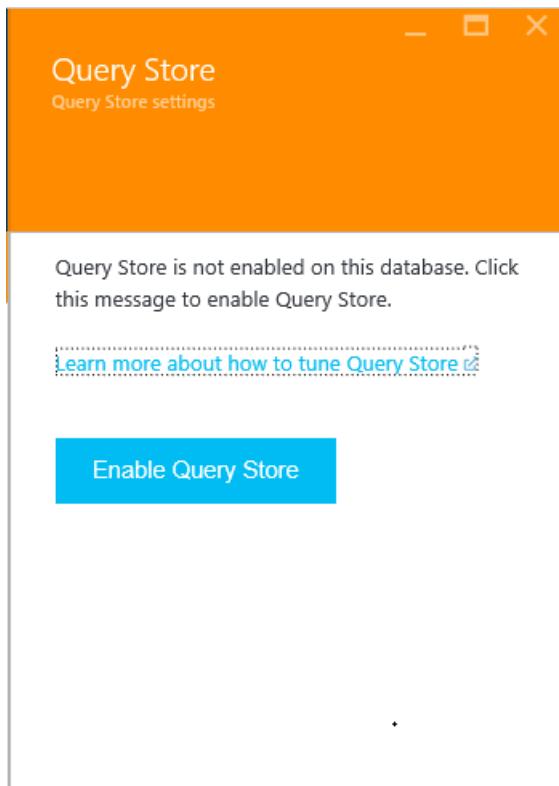
These messages usually appear when Query Store is not able to collect new data.

First case happens when Query Store is in Read-Only state and parameters are set optimally. You can fix this by increasing size of Query Store or clearing Query Store.



Second case happens when Query Store is Off or parameters aren't set optimally.

You can change the Retention and Capture policy and enable Query Store by executing commands below or directly from portal:



### Recommended retention and capture policy

There are two types of retention policies:

- Size based - if set to AUTO it will clean data automatically when near max size is reached.
- Time based - by default we will set it to 30 days, which means, if Query Store will run out of space, it will delete query information older than 30 days

Capture policy could be set to:

- **All** - Captures all queries.
- **Auto** - Infrequent queries and queries with insignificant compile and execution duration are ignored. Thresholds for execution count, compile and runtime duration are internally determined. This is the default option.
- **None** - Query Store stops capturing new queries, however runtime stats for already captured queries are still collected.

We recommend setting all policies to AUTO and clean policy to 30 days:

```
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 size of Query Store. This could be performed by connecting to a database and issuing following query:

```
ALTER DATABASE [YourDB]
SET QUERY_STORE (MAX_STORAGE_SIZE_MB = 1024);
```

Applying these settings will eventually make Query Store collecting new queries, however if you don't want to wait you can clear Query Store.

#### NOTE

Executing following query will delete all current information in the Query Store.

```
ALTER DATABASE [YourDB] SET QUERY_STORE CLEAR;
```

## Summary

Query Performance Insight helps you understand the impact of your query workload and how it relates to database resource consumption. With this feature, you will learn about the top consuming queries, and easily identify the ones to fix before they become a problem.

## Next steps

For additional recommendations about improving the performance of your SQL database, click [Recommendations](#) on the **Query Performance Insight** blade.



# Performance recommendations for SQL Database

9/24/2018 • 5 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 for performance tuning. [Intelligent Insights](#) is the recommended method for monitoring performance.

## 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

*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.                   |

## 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.

# Find and apply performance recommendations

10/2/2018 • 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]<br>Indexed columns: [index_1],[index_2],[index_3]                    | HIGH IMPACT |
| CREATE INDEX                | Table: [test_table_0.914675]<br>Indexed columns: [index_1],[index_2],[index_3]                    | HIGH IMPACT |
| DROP INDEX (PREVIEW)        | Index name: IR_[test_schema]_[test_table_0.112348]_CD2E5085881888FC9A4<br>Reason: Duplicate index | HIGH IMPACT |
| DROP INDEX (PREVIEW)        | Index name: IR_[test_schema]_[test_table_0.950691]_9A67D9E88A31B315D14<br>Reason: Duplicate index | HIGH IMPACT |
| FIX SCHEMA ISSUES (PREVIEW) | Error code: 208<br>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.

A screenshot of the Azure portal showing a 'Create index' recommendation for the [dbo].[Employees] table. The window title is 'Create index'. The top bar includes 'Apply', 'Discard', and 'View script' buttons. The main table has columns: RECOMMENDED ACTION, STATUS, LAST UPDATE, and INITIATED BY. A row shows 'Create index' (selected), 'Active', '5/19/2017 2:42:12 PM', and 'N/A'. Below the table are sections for 'Estimated impact' (Impact: HIGH, Disk space needed: 128.00 MB) and 'Details' (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**.



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**.

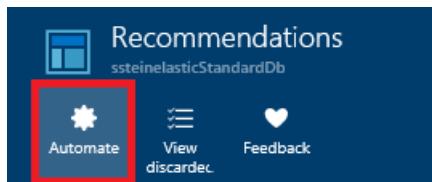
#### NOTE

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<br>Forced by user |
| CREATE INDEX | ON OFF INHERIT | OFF<br>Forced by user |
| DROP INDEX   | ON OFF INHERIT | OFF<br>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 run the recommended T-SQL script

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 \(Transact-SQL\)](#).

### 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 |
|--------|-------------|
|--------|-------------|

| 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:

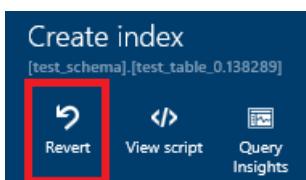
#### Tuning history

| ACTION  | RECOMMENDATION DESCRIPTION   | STATUS  | TIME                    |
|---|--|---------|-------------------------|
| DROP INDEX (PREVIEW)<br>Initiated by: User          | Index name: IR_[test_schema]_[test_table_0.182511]_1665B81581<br>Reason: Duplicate index   | Success | 4/25/2016<br>4:28:05 PM |
| CREATE INDEX<br>Initiated by: User                  | Table: [test_table_0.138289]<br>Indexed columns: [index_1],[index_2],[index_3]             | Success | 4/25/2016<br>4:27:57 PM |
| PARAMETERIZE QUERIES (PREVIEW)<br>Initiated by: N/A | Scope: Entire database<br>Reason: Non-parameterized queries are causing performance issues | Success | 4/21/2016<br>4:40:30 PM |
| DROP INDEX (PREVIEW)<br>Initiated by: User          | Index name: IR_[test_schema]_[test_table_0.574879]_C13F85C293<br>Reason: Duplicate index   | Error   | 4/25/2016<br>4:28:05 PM |

#### Reverting a recommendation

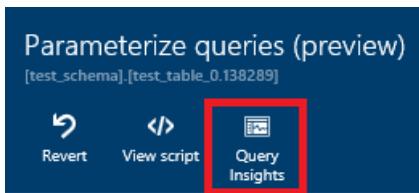
If you used the performance recommendations to apply the recommendation (meaning you did not manually run the T-SQL script), it automatically reverts the change if it finds the performance impact to be negative. If for any reason you simply want to revert a recommendation, you can do the following:

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

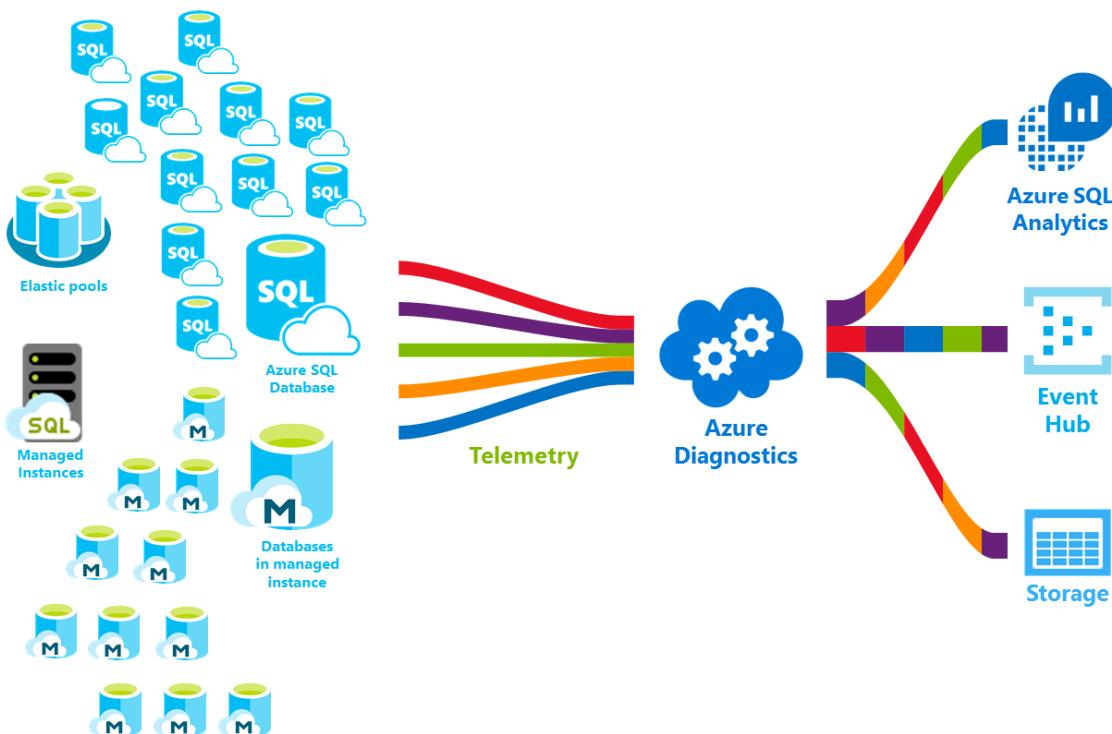
- [Query Store](#)
- [CREATE INDEX](#)
- [Role-based access control](#)

# Azure SQL Database metrics and diagnostics logging

10/25/2018 • 22 minutes to read • [Edit Online](#)

Azure SQL Database, elastic pools, Managed Instance, and databases in Managed Instance can emit metrics and diagnostics logs for easier performance monitoring. You can configure a database to stream resource usage, workers and sessions, and connectivity into one of these Azure resources:

- **Azure SQL Analytics:** Used as integrated Azure database intelligent performance monitoring solution with reporting, alerting, and mitigating capabilities.
- **Azure Event Hubs:** Used for integrating SQL Database telemetry with your custom monitoring solution or hot pipelines.
- **Azure Storage:** Used for archiving vast amounts of telemetry for a fraction of the price.



To understand the metrics and log categories that are supported by the various Azure services, you might want to consider reading:

- [Overview of metrics in Microsoft Azure](#)
- [Overview of Azure diagnostics logs](#)

## Enable logging of diagnostics telemetry

Use the first section of this document to enable diagnostics telemetry for databases, and the second part of the document to enable diagnostics telemetry for elastic pools, or Managed Instances. Use the later sections of this document to configure Azure SQL Analytics as a monitoring tool for viewing of the streamed database diagnostics telemetry.

#### NOTE

In case you are using elastic pools or Managed Instances, besides enabling diagnostics telemetry for databases, you are also advised to enable diagnostics telemetry for these resources as well. This is because elastic pools and Managed Instances in the role of database containers have its own diagnostics telemetry which is separate from an individual database 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 where selected data will be collected. Options available include:

- Azure SQL Analytics
- Azure Event Hubs
- Azure Storage

You can provision a new Azure resource or select an existing resource. After selecting a resource, using Diagnostic settings option, you need to specify which data to collect.

## Enable logging for Azure SQL Database or databases in Managed Instance

Metrics and diagnostics logging on SQL Database, and databases in Managed Instance are not enabled by default.

The following diagnostics telemetry is available for collection for Azure SQL Databases, and databases in Managed Instance:

| MONITORING TELEMETRY FOR DATABASES   | AZURE SQL DATABASE SUPPORT | DATABASE IN MANAGED INSTANCE SUPPORT |
|--|----------------------------|--------------------------------------|
| All 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                                   |
| QueryStoreRuntimeStatistics: Contains information about the query runtime statistics, such as CPU usage and query duration stats.  | Yes                        | Yes                                  |
| QueryStoreWaitStatistics: Contains information about the query wait statistics, which tells you what your queries waited on, such as CPU, LOG, and LOCKING.  | Yes                        | Yes                                  |

| MONITORING TELEMETRY FOR DATABASES   | AZURE SQL DATABASE SUPPORT | DATABASE IN MANAGED INSTANCE SUPPORT |
|--|----------------------------|--------------------------------------|
| <b>Errors:</b> Contains information about SQL errors that happened on this database.                                       | Yes                        | No                                   |
| <b>DatabaseWaitStatistics:</b> Contains information about how much time a database spent waiting on different wait types.  | Yes                        | No                                   |
| <b>Timeouts:</b> Contains information about timeouts that happened on a database.  | Yes                        | No                                   |
| <b>Blocks:</b> Contains information about blocking events that happened on a database.                                     | Yes                        | No                                   |
| <b>SQLInsights:</b> Contains Intelligent Insights into performance. <a href="#">Learn more about Intelligent Insights.</a> | Yes                        | Yes                                  |

## Azure portal

Streaming of diagnostics telemetry for Azure SQL Database, and databases in Managed Instance to destinations of Azure storage, event hubs, or Log Analytics is configured through Diagnostics settings menu for each of databases in Azure portal.

### Configure streaming of diagnostics telemetry for Azure SQL Database



To enable streaming of diagnostics telemetry for **Azure SQL Database**, follow these steps:

1. Go to your 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
4. Up to three (3) parallel connections to stream diagnostics telemetry can be created. To configure multiple parallel streaming of diagnostics data to multiple resources, select **+Add diagnostic setting** to create an additional setting.

The screenshot shows the 'Diagnostic settings' page for a SQL database named 'CRM Database'. The left sidebar lists several monitoring and management options. The 'Diagnostic settings' option is highlighted with a red box. The main pane displays a subscription ('Workload Insight dev/test subscription'), a resource group ('widemo'), and a resource type ('0 selected'). A prominent button at the bottom left says 'Turn on diagnostics'.

5. Type in the name for the setting - for your own reference
6. Select to which resource to stream diagnostics data from the database: **Archive to storage account, Stream to an event hub, or Send to Log Analytics**
7. For standard monitoring experience, select checkboxes for database diagnostics log telemetry: **SQLInsights, AutomaticTuning, QueryStoreRuntimeStatistics, QueryStoreWaitStatistics, Errors, DatabaseWaitStatistics, Timeouts, Blocks, Deadlocks**. This telemetry is event based and provides the standard monitoring experience.
8. For advanced monitoring experience, select checkbox for **AllMetrics**. This is a 1-minute based telemetry for the database diagnostics telemetry as described above.
9. Click on **Save**

## Diagnostics settings

X

Save  Discard  Delete

\* Name

service



Archive to a storage account

Stream to an event hub

Send to Log Analytics

Log Analytics  
Configure >

LOG

SQLInsights

AutomaticTuning

QueryStoreRuntimeStatistics

QueryStoreWaitStatistics

Errors

DatabaseWaitStatistics

Timeouts

Blocks

Deadlocks

Audit

SQLSecurityAuditEvents

METRIC

AllMetrics

### NOTE

Audit log cannot be enabled from database Diagnostics settings. To enable Audit log streaming, see [Set up auditing for your database](#), and also see [SQL Audit logs in Azure Log Analytics and Azure Event Hubs](#).

### TIP

Repeat the above steps for each Azure SQL Database you wish to monitor.

## Configure streaming of diagnostics telemetry for databases in Managed Instance



To enable streaming of diagnostics telemetry for **databases in Managed Instance**, follow these steps:

1. Go to your database in 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
4. Up to three (3) parallel connections to stream diagnostics telemetry can be created. To configure multiple parallel streaming of diagnostics data to multiple resources, select **+Add diagnostic setting** to create an additional setting.

A screenshot of the Azure portal showing the "CRM Database - Diagnostic settings" page. The left sidebar includes links for Overview, Activity log, Diagnose and solve problems, Settings (Locks, Automation script), Security (Advanced Threat Protection), Monitoring (Diagnostic settings), and Support + troubleshooting (New support request). The "Diagnostic settings" link is highlighted with a red box. The main pane 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". A red box highlights the "Turn on diagnostics" button, which is described as "to collect the following data".

5. Type in the name for the setting - for your own reference
6. Select to which resource to stream diagnostics data from the database: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**
7. Select checkboxes for database diagnostics telemetry: **SQLInsights**, **QueryStoreRuntimeStatistics**, **QueryStoreWaitStatistics** and **Errors**
8. Click on **Save**

## Diagnostics settings

Save  Discard  Delete

\* Name

service

X

Archive to a storage account

Stream to an event hub

Send to Log Analytics

Log Analytics  
Configure >

LOG

SQLInsights

QueryStoreRuntimeStatistics

QueryStoreWaitStatistics

Errors

### TIP

Repeat the above steps for each database in Managed Instance you wish to monitor.

## Enable logging for elastic pools or Managed Instance

Elastic pools and Managed Instances as database containers have its own diagnostics telemetry separate from databases. This diagnostics telemetry is not enabled by default.

### Configure streaming of diagnostics telemetry for elastic pools



Elastic pools

The following diagnostics telemetry is available for collection for elastic pools resource:

| RESOURCE     | MONITORING TELEMETRY  |
|--------------|---|
| Elastic pool | All 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 enable streaming of diagnostics telemetry for **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

The screenshot shows the 'Diagnostic settings' page for an elastic pool named 'epdemoaton'. The left sidebar contains links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Quick start, Configure, Locks, Automation script), Monitoring (Alerts (Classic), Database Resource Utilization, Metrics), and Support + troubleshooting (Recommendations, New support request). The 'Diagnostic settings' link is highlighted with a red box. The main content area shows a 'Turn on diagnostics' button with a red box around it, indicating where to click to start collecting data.

4. Type in the name for the setting - for your own reference
5. Select to which resource to stream diagnostics data from the elastic pool: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**
6. In case Log Analytics is selected, select **Configure** and create a new workspace by selecting **+Create New Workspace**, or select an existing workspace
7. Select the checkbox for elastic pool diagnostics telemetry **AllMetrics**
8. Click **Save**

The screenshot shows the 'Diagnostics settings' configuration dialog. It includes fields for Name (set to 'service'), Archive to a storage account (unchecked), Stream to an event hub (unchecked), Send to Log Analytics (checked), and a Log Analytics Configure button. Below these, there is a METRIC section with an AllMetrics checkbox (checked) highlighted with a red box.

## TIP

Repeat the above steps for each elastic pool you wish to monitor.

## Configure streaming of diagnostics telemetry for Managed Instance



The following diagnostics telemetry is available for collection for Managed Instance resource:

| RESOURCE                | MONITORING TELEMETRY   |
|-------------------------|--|
| <b>Managed Instance</b> | <a href="#">ResourceUsageStats</a> contains vCores count, average CPU percentage, IO requests, bytes read/written, reserved storage space, used storage space. |

To enable streaming of diagnostics telemetry for **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

The screenshot shows the 'CRM Database - Diagnostic settings' page. On the left, there's a sidebar with links like Overview, Activity log, and Diagnostic settings (which is highlighted with a red box). The main area shows subscription details (Workload Insight dev/test subscription), resource group (widemo), and resource type (0 selected). A prominent red box highlights the 'Turn on diagnostics' button, which is described as 'to collect the following data.'

4. Type in the name for the setting - for your own reference
5. Select to which resource to stream diagnostics data from the elastic pool: **Archive to storage account**, **Stream to an event hub**, or **Send to Log Analytics**
6. In case Log Analytics is selected, create or use an existing workspace
7. Select the checkbox for instance diagnostics telemetry **ResourceUsageStats**
8. Click **Save**

## Diagnostics settings

Save Discard Delete

\* Name

service

X

Archive to a storage account

Stream to an event hub

Send to Log Analytics

Log Analytics  
Configure >

LOG

SQLInsights

QueryStoreRuntimeStatistics

QueryStoreWaitStatistics

Errors

### TIP

Repeat the above steps for each Managed Instance you wish to monitor.

## PowerShell

To enable metrics and diagnostics logging by using PowerShell, use the following commands:

- To enable storage of diagnostics logs in a storage account, use this command:

```
Set-AzureRmDiagnosticSetting -ResourceId [your resource id] -StorageAccountId [your storage account id]  
-Enabled $true
```

The storage account ID is the resource ID for the storage account where you want to send the logs.

- To enable streaming of diagnostics logs to an event hub, use this command:

```
Set-AzureRmDiagnosticSetting -ResourceId [your resource id] -ServiceBusRuleId [your service bus rule  
id] -Enabled $true
```

The Azure Service Bus rule ID is a string with this format:

```
{service bus resource ID}/authorizationrules/{key name}
```

- To enable sending diagnostics logs to a Log Analytics workspace, use this command:

```
Set-AzureRmDiagnosticSetting -ResourceId [your resource id] -WorkspaceId [resource id of the log analytics workspace] -Enabled $true
```

- You can obtain the resource ID of your Log Analytics workspace by using the following command:

```
(Get-AzureRmOperationalInsightsWorkspace).ResourceId
```

You can combine these parameters to enable multiple output options.

### To configure multiple Azure resources

To support multiple subscriptions, use the PowerShell script from [Enable Azure resource metrics logging using PowerShell](#).

Provide the workspace resource ID <\$WSID> as a parameter when executing the script (Enable-AzureRMDiagnostics.ps1) to send diagnostic data from multiple resources to the workspace. To get the workspace ID <\$WSID> to which you would like to send diagnostic data, replace <subID> with the subscription ID, replace <RG\_NAME> with the resource group name, and replace <WS\_NAME> with the workspace name in the following script.

- To configure multiple Azure resources, use the following commands:

```
PS C:\> $WSID =
"/subscriptions/<subID>/resourcegroups/<RG_NAME>/providers/microsoft.operationalinsights/workspaces/<WS_NAME>"
PS C:\> .\Enable-AzureRMDiagnostics.ps1 -WSID $WSID
```

### Azure CLI

To enable metrics and diagnostics logging by using the Azure CLI, use the following commands:

- To enable storage of diagnostics logs in a storage account, use this command:

```
azure insights diagnostic set --resourceId <resourceId> --storageId <storageAccountId> --enabled true
```

The storage account ID is the resource ID for the storage account where you want to send the logs.

- To enable streaming of diagnostics logs to an event hub, use this command:

```
azure insights diagnostic set --resourceId <resourceId> --serviceBusRuleId <serviceBusRuleId> --enabled true
```

The Service Bus rule ID is a string with this format:

```
{service bus resource ID}/authorizationrules/{key name}
```

- To enable sending diagnostics logs to a Log Analytics workspace, use this command:

```
azure insights diagnostic set --resourceId <resourceId> --workspaceId <resource id of the log analytics workspace> --enabled true
```

You can combine these parameters to enable multiple output options.

### REST API

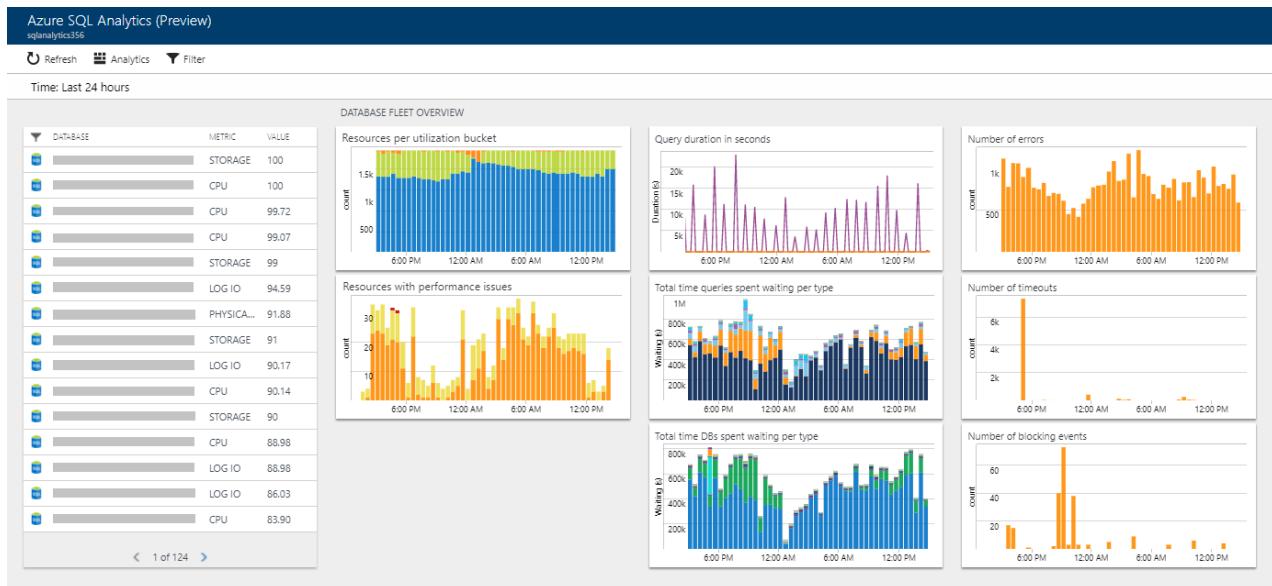
Read about how to [change diagnostics settings by using the Azure Monitor REST API](#).

## Resource Manager template

Read about how to [enable diagnostics settings at resource creation by using a Resource Manager template](#).

# Stream into Azure SQL Analytics

Azure SQL Analytics is a cloud monitoring solution for monitoring performance of Azure SQL databases, elastic pools, and Managed Instances at scale and across multiple subscriptions through a single pane of glass. It collects and visualizes important Azure SQL Database performance metrics with 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 blade in the portal. You also can enable Log Analytics by using a diagnostics setting via PowerShell cmdlets, the Azure CLI, or the Azure Monitor REST API.

## Installation overview

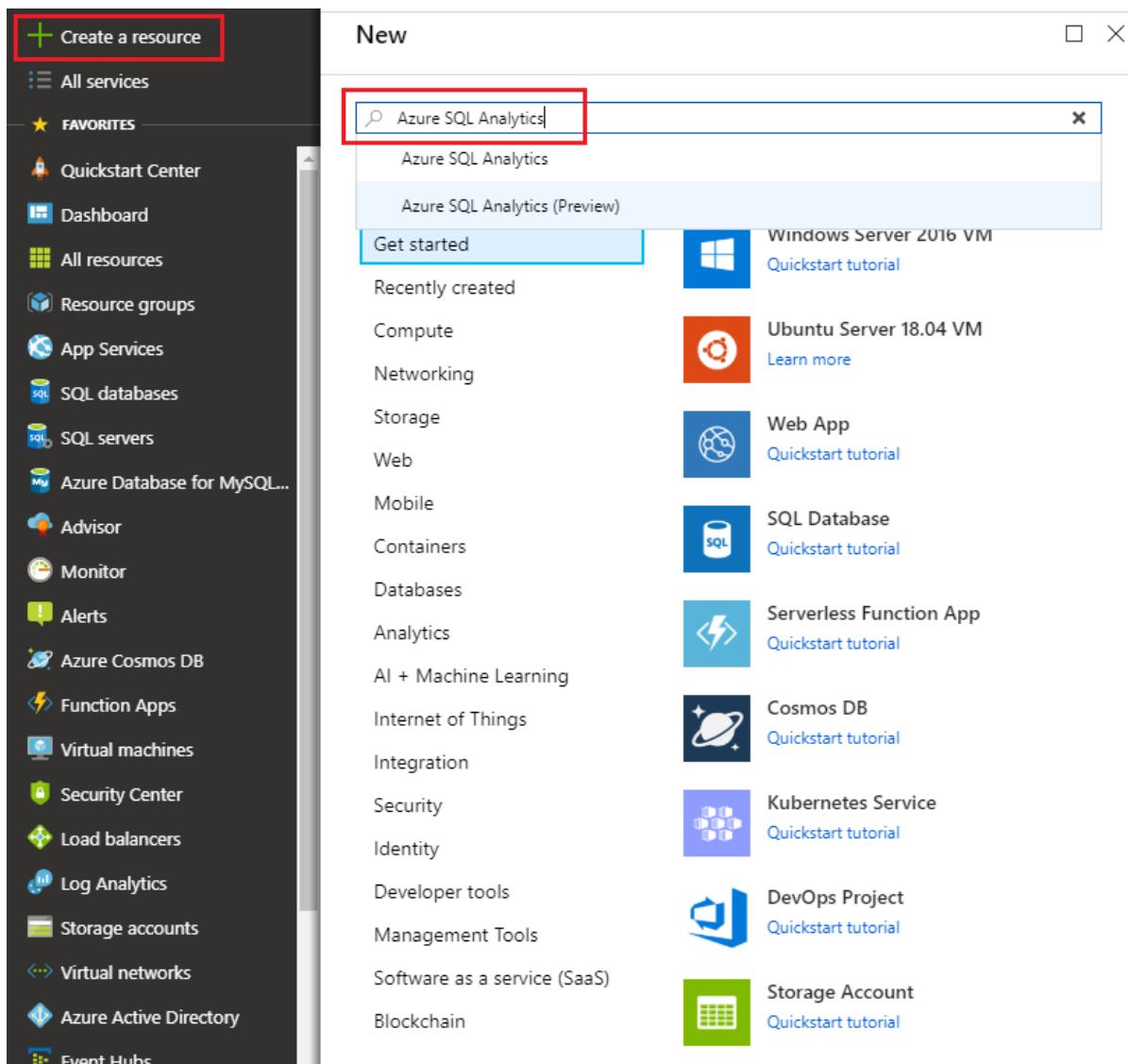
Monitoring a SQL Database fleet is simple with Azure SQL Analytics. Three steps are required:

1. Create Azure SQL Analytics solution from the Azure Marketplace
2. Create monitoring workspace in the solution
3. Configure databases to stream diagnostics telemetry into the workspace you've created.

In case you are using elastic pools or Managed Instances, in addition to configuring database diagnostics telemetry, configure streaming of diagnostics telemetry from these resources as well.

## Create 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 screenshot shows the Azure portal interface for creating an OMS workspace. The top navigation bar includes 'Home', 'New', 'Azure SQL Analytics (Preview)', 'Azure SQL Analytics (Preview)', 'OMS Workspaces', 'Log analytics workspace', and 'Pricing Tier'. The main area displays a list of existing OMS workspaces: cnfwtest (eastus), CostConsumptionMonitor (westeurope), DaniLogsOnly (westeurope), DaniNoMetrics (westeurope), DaniWS (westeurope), DefaultBGFSWorkspace (westeurope), and DefaultWorkspace-741fd... (centralindia). A red box highlights the 'Create New Workspace' button. To the right, a 'Log analytics workspace' dialog box is open, showing fields for 'OMS Workspace' (name: 'sqlanalytics356'), 'Subscription' (selected: 'BASE\_BizAnalyz\_10137371\_mwories'), 'Resource group' (selected: 'Create new'), 'Location' (selected: 'East US'), and 'Pricing tier' (selected: 'Free'). A red box highlights the 'Pricing tier' dropdown. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

4. Confirm by selecting **OK**, and finalize by selecting **Create**

#### Configure databases to record metrics and diagnostics logs

The easiest way to configure where databases record their metrics is through the Azure portal - as described above. In the portal, go to your SQL Database resource and select **Diagnostics settings**.

In the case you are using elastic pools or Managed Instances, you will also need to configure Diagnostics settings in these resources as well to stream their own diagnostics telemetry into the workspace you've created.

## Use the SQL Analytics solution

SQL Analytics is a hierarchical dashboard that allows you to move through the hierarchy of SQL Database resources. To learn how to use the SQL Analytics solution, see [Monitor SQL Database by using the SQL Analytics solution](#).

## Stream into Event Hubs

SQL Database metrics and diagnostics logs can be streamed into Event Hubs by using the built-in **Stream to an event hub** option in the portal. You also can enable the Service Bus rule ID by using a diagnostics setting 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 any real-time analytics provider or batching/storage adapters. 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](#)

Here are a few ways that you might use the streaming capability:

- **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 different third-party monitoring and log analytics solutions.
- **Build a custom telemetry and logging platform.** If 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 into Storage

SQL Database metrics and diagnostics logs can be stored in Storage by using the built-in **Archive to a storage account** option in the portal. You also can enable Storage by using a diagnostics setting 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 these 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=00/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 all 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
```

If you want to record the data from the elastic pool, the blob name is a bit different:

```
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
```

## Download metrics and logs from Storage

Learn how to [download metrics and diagnostics logs from Storage](#).

## 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. Consumption of diagnostics telemetry above the free units of data ingestion allocated each month applies. The free units of data ingestion provided enable free monitoring of several databases each month. Please note that more active databases with heavier workloads will ingest more data versus idle databases. For more information, see [Log Analytics pricing](#).

If you are using Azure SQL Analytics, you can easily monitor your data ingestion consumption in the solution by selecting OMS Workspace on the navigation menu of Azure SQL Analytics, and then selecting Usage and Estimated Costs.

## Metrics and logs available

Collected monitoring telemetry can be used for your own **custom analysis** and **application development** using [SQL Analytics language](#). Structure of the collected data, metrics and logs, is listed below.

## All metrics

### All 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 |

### All metrics for Azure SQL Database

| RESOURCE | METRICS |
|----------|---------|
|          |         |

| RESOURCE           | METRICS  |
|--------------------|--|
| Azure SQL 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 |

## Logs

### Logs for Managed Instance

#### Resource Usage Stats

| 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 that the database belongs to.          |
| ResourceGroup           | Name of the resource group that the database belongs to. |
| LogicalServerName_s     | Name of the Managed Instance.                            |
| ResourceId              | Resource URI.  |
| SKU_s                   | Managed Instance product SKU                             |
| virtual_core_count_s    | Numver of vCores available                               |
| avg_cpu_percent_s       | Average CPU percentage                                   |
| reserved_storage_mb_s   | Reserved storage capacity on Managed Instance            |
| storage_space_used_mb_s | Used storage on Managed Instance                         |
| io_requests_s           | IOPS count   |
| io_bytes_read_s         | IOPS bytes read  |

| PROPERTY           | DESCRIPTION        |
|--------------------|--------------------|
| io_bytes_written_s | IOPS bytes written |

## Logs for Azure SQL Database and Managed Instance database

### 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   |
| 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 that the database belongs to.                        |
| ResourceGroup          | Name of the resource group that the database belongs to.               |
| LogicalServerName_s    | Name of the server that the database belongs to.                       |
| ElasticPoolName_s      | Name of the elastic pool that the database belongs to, 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.   |

| PROPERTY                    | DESCRIPTION  |
|-----------------------------|--|
| 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.                           |
| 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                            |
| OperationName              | Name of the operation. Always: QueryStoreWaitStatisticsEvent                      |
| Resource                   | Name of the resource  |
| ResourceType               | Name of the resource type. Always: SERVERS/DATABASES                              |
| SubscriptionId             | Subscription GUID that the database belongs to.                                   |
| ResourceGroup              | Name of the resource group that the database belongs to.                          |
| LogicalServerName_s        | Name of the server that the database belongs to.                                  |
| ElasticPoolName_s          | Name of the elastic pool that the database belongs to, 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.  |

| PROPERTY               | DESCRIPTION  |
|------------------------|--|
| 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.  |
| 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 that the database belongs to.                |
| ResourceGroup       | Name of the resource group that the database belongs to.       |
| LogicalServerName_s | Name of the server that the database belongs to.               |
| ElasticPoolName_s   | Name of the elastic pool that the database belongs to, if any. |
| DatabaseName_s      | Name of the database.  |
| ResourceId          | Resource URI.  |

| PROPERTY          | DESCRIPTION  |
|-------------------|--|
| 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 that the database belongs to.                |
| ResourceGroup       | Name of the resource group that the database belongs to.       |
| LogicalServerName_s | Name of the server that the database belongs to.               |
| ElasticPoolName_s   | Name of the elastic pool that the database belongs to, if any. |
| DatabaseName_s      | Name of the database.  |
| ResourceId          | Resource URI.  |
| wait_type_s         | Name of the wait type.   |

| PROPERTY                    | DESCRIPTION                    |
|-----------------------------|--------------------------------|
| 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 signal 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 that the database belongs to.                |
| ResourceGroup       | Name of the resource group that the database belongs to.       |
| LogicalServerName_s | Name of the server that the database belongs to.               |
| ElasticPoolName_s   | Name of the elastic pool that the database belongs to, if any. |
| DatabaseName_s      | Name of the database.  |
| ResourceId          | Resource URI.  |
| error_state_d       | Error state code.  |
| query_hash_s        | Query hash, if available.                                      |

| PROPERTY          | DESCRIPTION                    |
|-------------------|--------------------------------|
| 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                           |
| OperationName              | Name of the operation. Always: BlockEvent                      |
| Resource                   | Name of the resource   |
| ResourceType               | Name of the resource type. Always: SERVERS/DATABASES           |
| SubscriptionId             | Subscription GUID that the database belongs to.                |
| ResourceGroup              | Name of the resource group that the database belongs to.       |
| LogicalServerName_s        | Name of the server that the database belongs to.               |
| ElasticPoolName_s          | Name of the elastic pool that the database belongs to, 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   |

| PROPERTY            | DESCRIPTION  |
|---------------------|--|
| 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 that the database belongs to.                |
| ResourceGroup       | Name of the resource group that the database belongs to.       |
| LogicalServerName_s | Name of the server that the database belongs to.               |
| ElasticPoolName_s   | Name of the elastic pool that the database belongs to, 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 that the database belongs to.      |

| PROPERTY              | DESCRIPTION  |
|-----------------------|--|
| ResourceGroup         | Name of the resource group that the database belongs to.       |
| LogicalServerName_s   | Name of the server that the database belongs to.               |
| LogicalDatabaseName_s | Name of the database.  |
| ElasticPoolName_s     | Name of the elastic pool that the database belongs to, 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.  |
| 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 understand the metrics and log categories supported by the various Azure services, read:

- [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 more about Storage, see how to [download metrics and diagnostics logs from Storage](#).

# Intelligent Insights using AI to monitor and troubleshoot database performance

10/29/2018 • 9 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:

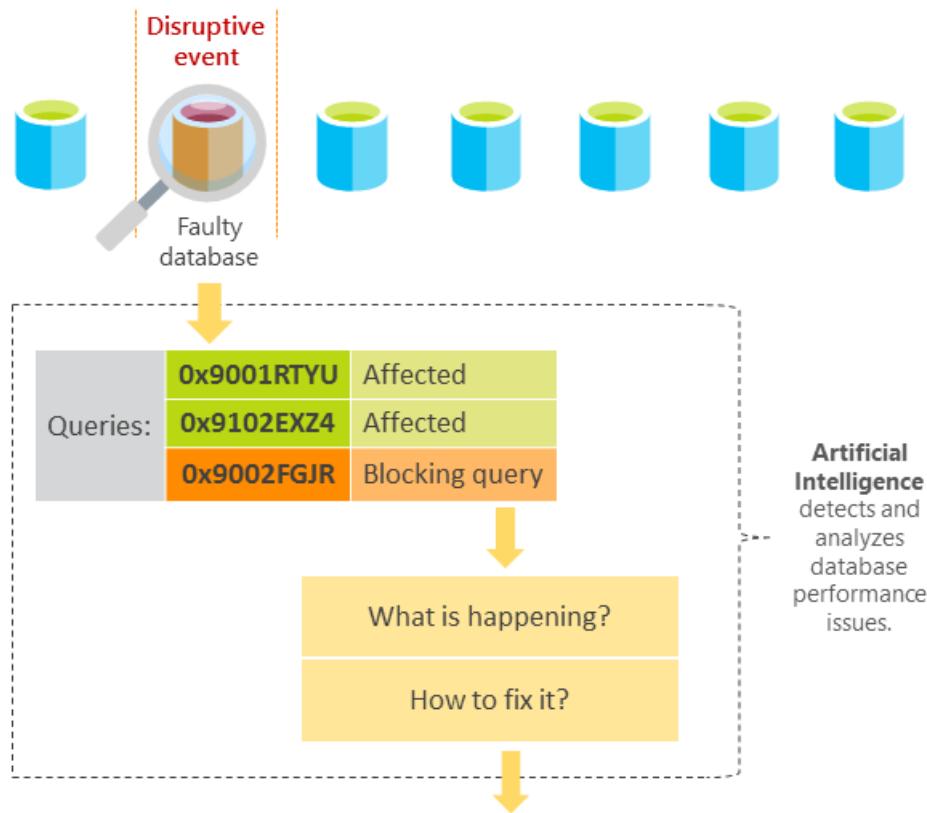
- Proactive monitoring
- Tailored performance insights
- Early detection of database performance degradation
- Root cause analysis of issues detected
- Performance improvement recommendations
- Scale out capability on hundreds of thousands of databases
- Positive impact to DevOps resources and the total 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. Updates are provided in the diagnostics log every 15 minutes.

## 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"><li>• Query duration increase [seconds].</li><li>• Excessive waiting [seconds].</li><li>• Timed-out requests [percentage].</li><li>• Errored-out requests [percentage].</li></ul> |
| 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 <a href="#">Troubleshoot database performance issues with Intelligent Insights</a> . |
| 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](#).

## 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.

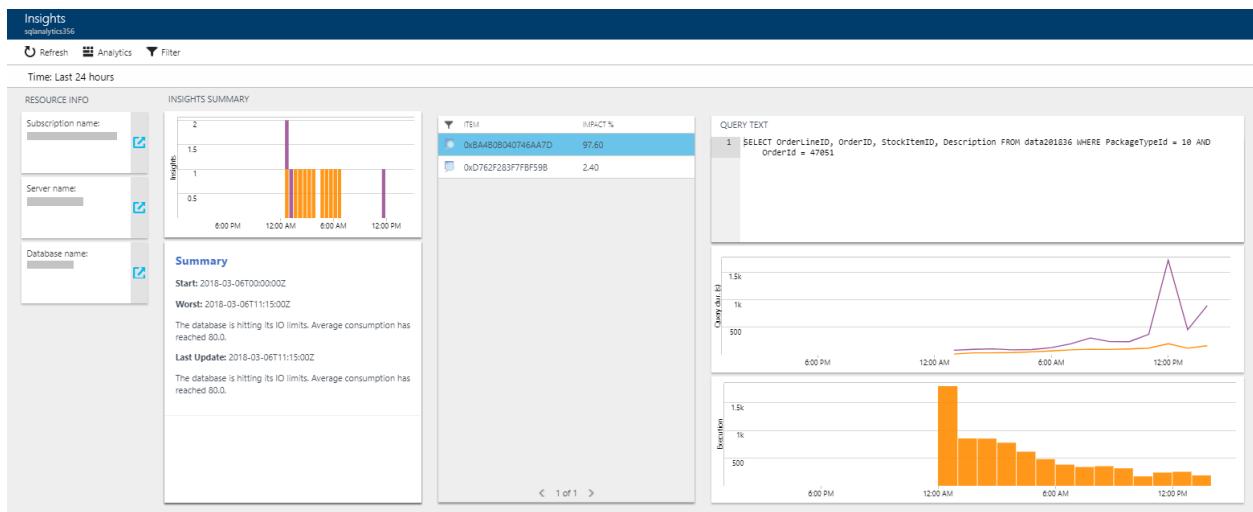
## TIP

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-requisite 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
- Errored 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](#).

# Troubleshoot Azure SQL Database performance issues with Intelligent Insights

10/16/2018 • 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 Log Analytics](#), [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-SScoped 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 logical 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.

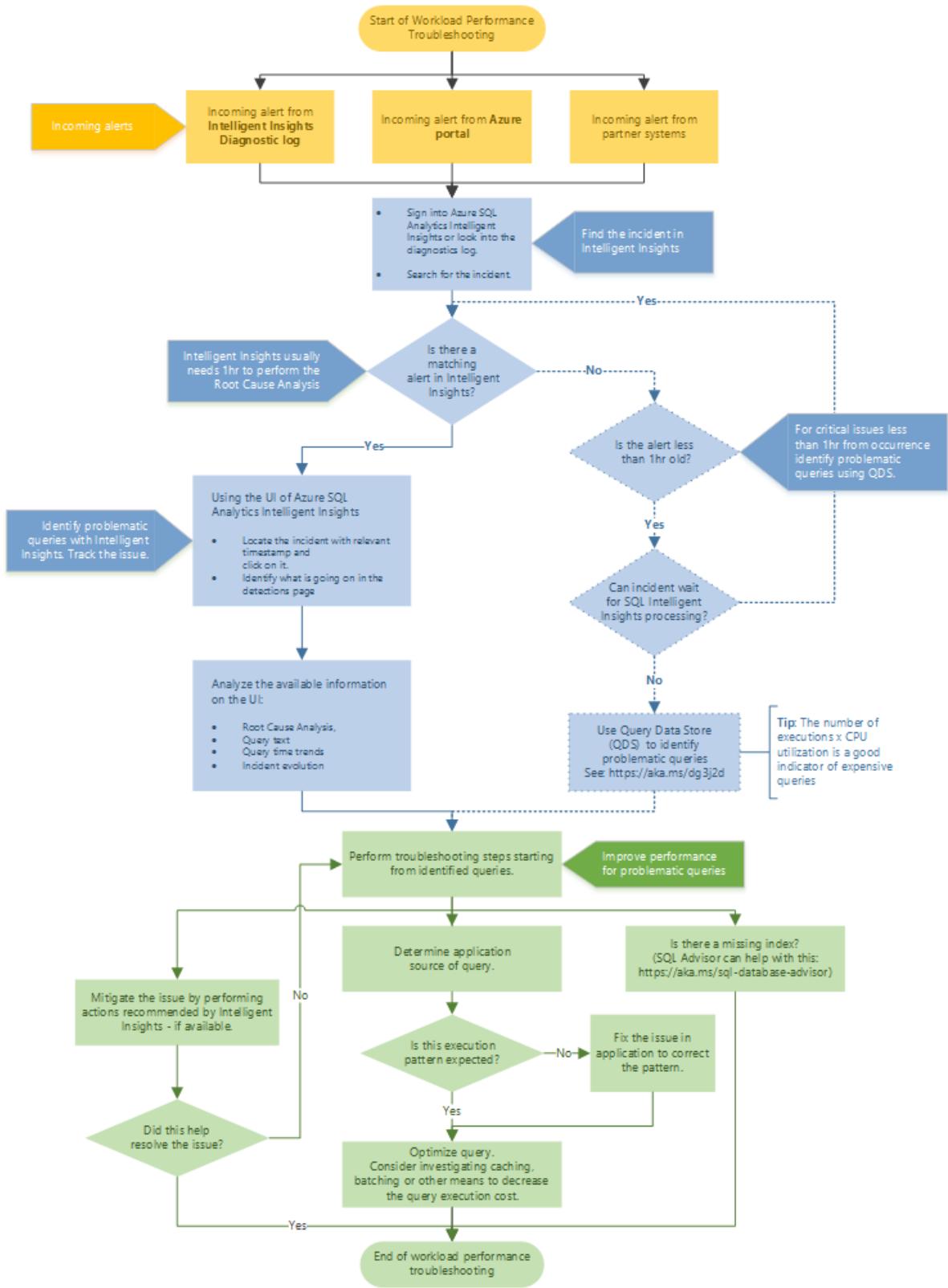
### Troubleshooting

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

9/25/2018 • 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 Log Analytics](#), [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 they're resolved. Intelligent Insights observes each issue lifecycle as "Active", "Verifying", or "Complete". Through each of these status phases, Intelligent Insights can record multiple event records in the log. For each of these entries, the issue ID number remains unique. Intelligent Insights tracks the issue through its lifecycle and generates an insight in the diagnostics log every 15 minutes.

After a performance issue is detected and for as long as it lasts, the issue is reported as "Active" under the status (status\_s) property. After a detected issue is mitigated, it's verified and reported as "Verifying" under the status (status\_s) property. If the issue is no longer present, the status (status\_s) property reports this issue as "Complete".

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"> <li>• Resources affected</li> <li>• Query hashes</li> <li>• Resource consumption percentage</li> </ul>   |
| Workload Increase               | <ul style="list-style-type: none"> <li>• Number of queries whose execution increased</li> <li>• Query hashes of queries with the largest contribution to the workload increase</li> </ul> |
| Memory Pressure                 | <ul style="list-style-type: none"> <li>• Memory clerk</li> </ul>  |
| Locking                         | <ul style="list-style-type: none"> <li>• Affected query hashes</li> <li>• Blocking query hashes</li> </ul>  |
| Increased MAXDOP                | <ul style="list-style-type: none"> <li>• Query hashes</li> <li>• CXP wait times</li> <li>• Wait times</li> </ul>  |
| Pagelatch Contention            | <ul style="list-style-type: none"> <li>• Query hashes of queries causing contention</li> </ul>  |
| Missing Index                   | <ul style="list-style-type: none"> <li>• Query hashes</li> </ul>  |

| DETECTABLE PERFORMANCE PATTERNS            | DETAILS OUTPUTTED  |
|--|--|
| New Query                                  | <ul style="list-style-type: none"> <li>Query hash of the new queries</li> </ul>  |
| Unusual Wait Statistic                     | <ul style="list-style-type: none"> <li>Unusual wait types</li> <li>Query hashes</li> <li>Query wait times</li> </ul>   |
| TempDB Contention                          | <ul style="list-style-type: none"> <li>Query hashes of queries causing contention</li> <li>Query attribution to the overall database pagelatch contention wait time [%]</li> </ul> |
| Elastic pool DTU Shortage                  | <ul style="list-style-type: none"> <li>Elastic pool</li> <li>Top DTU-consuming database</li> <li>Percent of pool DTU used by the top consumer</li> </ul>                           |
| Plan Regression                            | <ul style="list-style-type: none"> <li>Query hashes</li> <li>Good plan IDs</li> <li>Bad plan IDs</li> </ul>  |
| Database-Spaced Configuration Value Change | <ul style="list-style-type: none"> <li>Database-scoped configuration changes compared to the default values</li> </ul>   |
| Slow Client                                | <ul style="list-style-type: none"> <li>Query hashes</li> <li>Wait times</li> </ul>   |
| Pricing Tier Downgrade                     | <ul style="list-style-type: none"> <li>Text notification</li> </ul>  |

## 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 Log Analytics](#) 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](#).

# Automatic tuning in Azure SQL Database

10/2/2018 • 4 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.

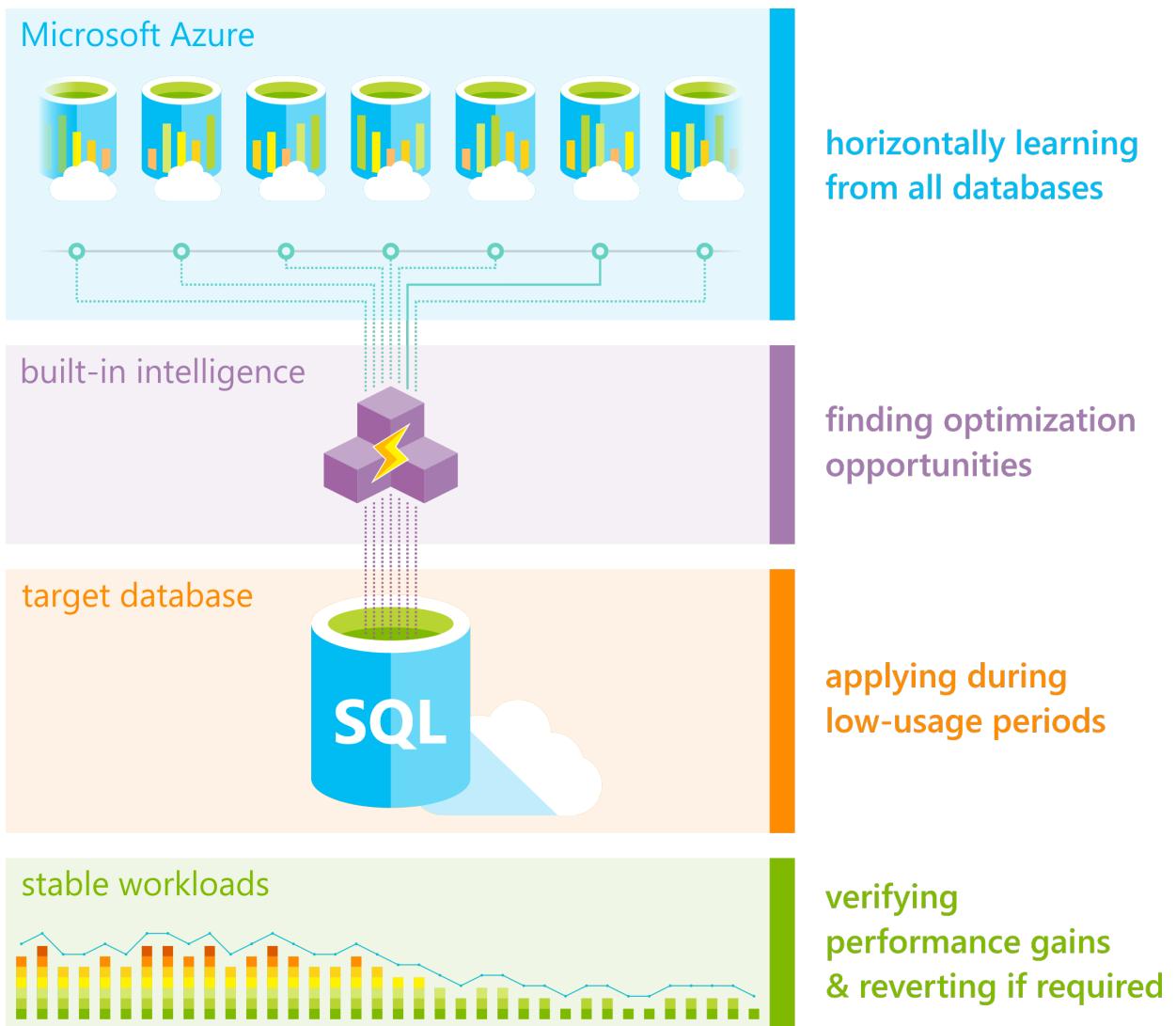
## What can Automatic Tuning do for you?

- 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 manually 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:

1. **CREATE INDEX** - identifies indexes that may improve performance of your workload, creates indexes, and automatically verifies that performance of queries has improved.

2. **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 at this time the option is not compatible with applications using partition switching and index hints.
3. **FORCE LAST GOOD PLAN** - 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.

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](#).

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 in such case it automatically validates there exists a positive gain to the workload performance, or otherwise if a regression is 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 manually applying tuning recommendations, the automatic performance validation, and reversal mechanisms are not available.

Automatic tuning options can be independently enabled or disabled per database, or they can be configured on logical servers and applied on every database that inherits settings from the server. Logical 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.

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](#).

# Enable automatic tuning to monitor queries and improve workload performance

10/30/2018 • 3 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**.

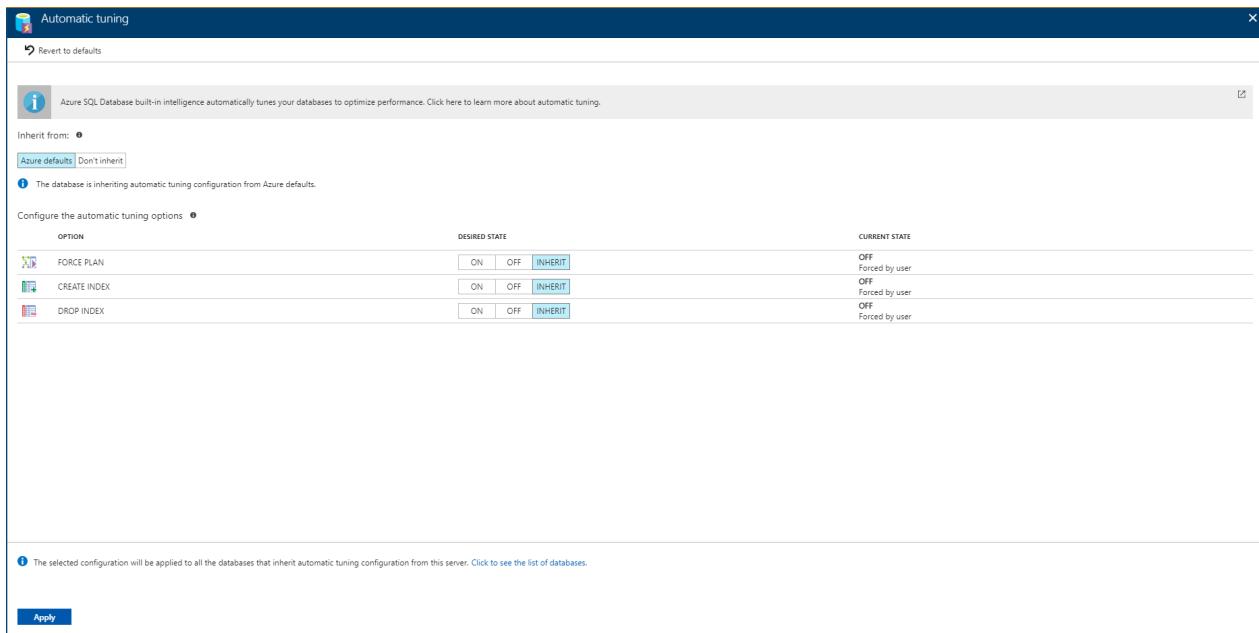
Automatic tuning can be enabled at the server or the database level through the [Azure portal](#), [REST API](#) calls and [T-SQL](#) commands.

## 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<br>Forced by user |
| CREATE INDEX | ON OFF INHERIT | ON<br>Forced by user  |
| DROP INDEX   | ON OFF INHERIT | OFF<br>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.

### 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.

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.

### NOTE

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.

| OPTION       | DESIRED STATE  | CURRENT STATE               |
|--------------|----------------|-----------------------------|
| FORCE PLAN   | ON OFF INHERIT | ON<br>Inherited from server |
| CREATE INDEX | ON OFF INHERIT | ON<br>Inherited from server |
| DROP INDEX   | ON OFF INHERIT | ON<br>Forced by user        |

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 configuration from the database level automatic tuning setting.

Find our more about T-SQL options to configure Automatic tuning, see [ALTER DATABASE SET Options \(Transact-SQL\) for SQL Database logical 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.

## 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

10/19/2018 • 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.

## 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

The screenshot shows the Azure Marketplace search interface. On the left, there is a sidebar with categories: Everything, Compute, Networking, Storage, Web + Mobile, Databases, Data + Analytics, and AI + Cognitive Services. The main area has a search bar with 'automation' typed in. Below the search bar is a 'Filter' button. The results table has columns for NAME, PUBLISHER, and CATEGORY. There are three items listed:

| 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.

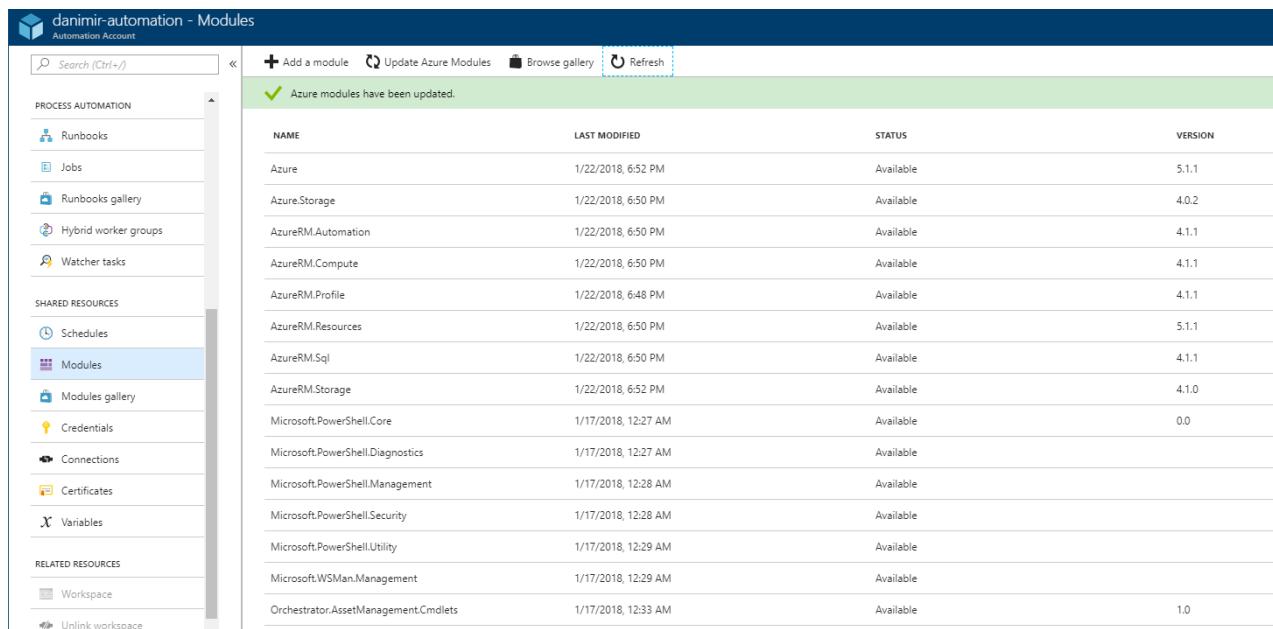
If you have several Azure subscriptions for which you would like to build the same automation, you need to repeat this process for your other subscriptions.

## Update Azure Automation modules

The PowerShell script to retrieve Automatic tuning recommendation uses [Get-AzureRmResource](#) and [Get-AzureRmSqlDatabaseRecommendedAction](#) commands for which update of Azure Modules to the version 4 and above is required.

Follow these steps to update Azure PowerShell modules:

- Access the Automation app pane, and select “**Modules**” on the left-hand side menu (scroll down as this menu item is under Shared Resources).
- In the Modules pane, click on “**Update Azure Modules**” at the top, and wait until the message “Azure modules have been updated” is displayed. This process can take a few minutes to complete.



The screenshot shows the Azure Automation Modules page. On the left, there's a sidebar with options like Runbooks, Jobs, and Schedules. The 'Modules' option is selected. At the top, there's a search bar and buttons for 'Add a module', 'Update Azure Modules' (which is highlighted with a blue border), 'Browse gallery', and 'Refresh'. A green banner at the top says 'Azure modules have been updated.' Below is a table listing various Azure modules with their names, last modified dates, statuses, and versions.

| NAME                                 | LAST MODIFIED       | STATUS    | VERSION |
|--------------------------------------|---------------------|-----------|---------|
| Azure                                | 1/22/2018, 6:52 PM  | Available | 5.1.1   |
| Azure.Storage                        | 1/22/2018, 6:50 PM  | Available | 4.0.2   |
| AzureRM.Automation                   | 1/22/2018, 6:50 PM  | Available | 4.1.1   |
| AzureRM.Compute                      | 1/22/2018, 6:50 PM  | Available | 4.1.1   |
| AzureRM.Profile                      | 1/22/2018, 6:48 PM  | Available | 4.1.1   |
| AzureRM.Resources                    | 1/22/2018, 6:50 PM  | Available | 5.1.1   |
| AzureRM.Sql                          | 1/22/2018, 6:50 PM  | Available | 4.1.1   |
| AzureRM.Storage                      | 1/22/2018, 6:52 PM  | Available | 4.1.0   |
| Microsoft.PowerShell.Core            | 1/17/2018, 12:27 AM | Available | 0.0     |
| Microsoft.PowerShell.Diagnostics     | 1/17/2018, 12:27 AM | Available |         |
| Microsoft.PowerShell.Management      | 1/17/2018, 12:28 AM | Available |         |
| Microsoft.PowerShell.Security        | 1/17/2018, 12:28 AM | Available |         |
| Microsoft.PowerShell.Utility         | 1/17/2018, 12:29 AM | Available |         |
| Microsoft.WSMAN.Management           | 1/17/2018, 12:29 AM | Available |         |
| Orchestrator.AssetManagement.Cmdlets | 1/17/2018, 12:33 AM | Available | 1.0     |

Required version of AzureRM.Resources and AzureRM.Sql modules needs to be version 4 and above.

## Create Azure Automation Runbook

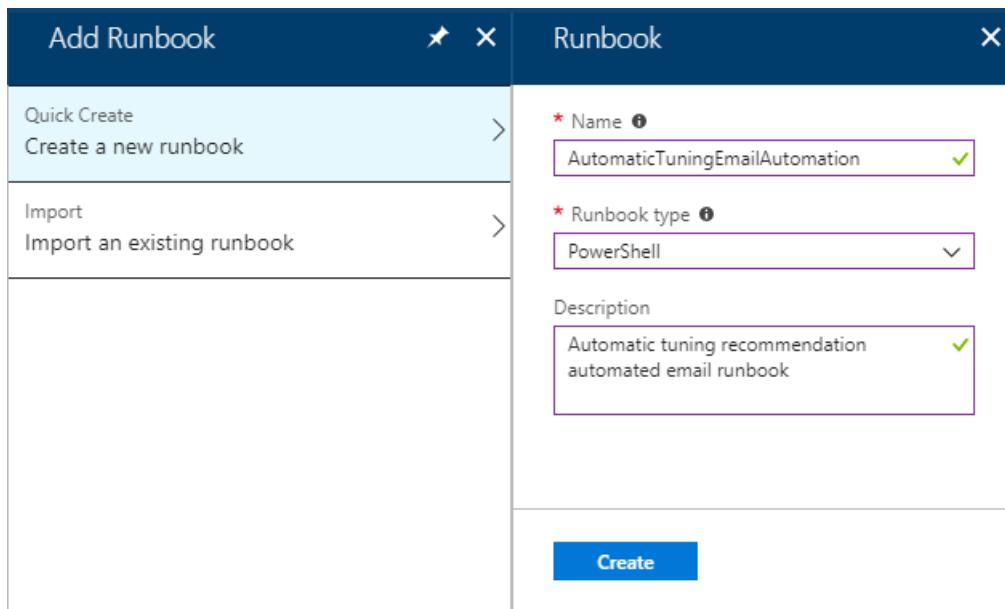
The next step is to create a Runbook in Azure Automation inside which the PowerShell script for retrieval of tuning recommendations resides.

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-AzureRmAccount -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-AzureRmSubscription -SubscriptionId $subscriptionId
    $rgs = Get-AzureRmResourceGroup

    # Loop through all resource groups
    foreach($rg in $rgs) {
        $rgname = $rg.ResourceGroupName;

        # Loop through all resource types
        foreach($resourceType in $resourceTypes) {
            $resources = Get-AzureRmResource -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-AzureRmSqlDatabaseRecommendedAction -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
$rg.ServerName
                            $object | Add-Member -Name 'DatabaseName' -MemberType NoteProperty -Value
$rg.DatabaseName
                            $object | Add-Member -Name 'Script' -MemberType NoteProperty -Value
$rg.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 Runbook interface. On the left, the 'Job' pane displays basic job details like Job ID, Created date, Last Update, Run As User, and Run on Azure. On the right, the 'Output' pane shows the PowerShell script's execution results. The script details include environments, connection parameters (Name, Account, Environment, Subscription, Tenant, TokenCache, VersionProfile), and three specific database index operations: dropping an index on 'test\_hinted\_drop\_name', creating a nonclustered index on 'DataPoints', and creating a nonclustered index on 'Employees'.

```
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

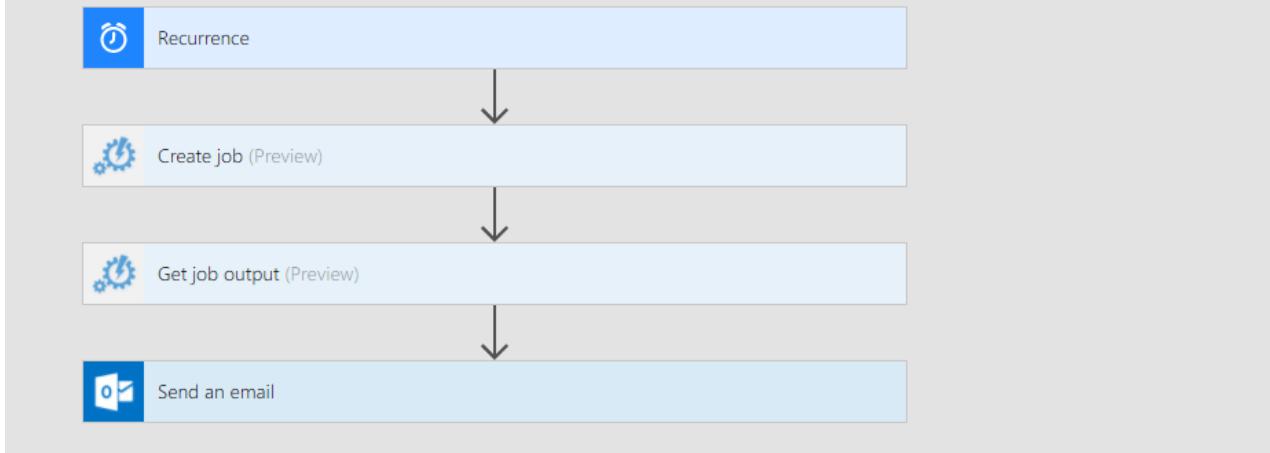
accomplish adding the required jobs to the flow, follow these steps:

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

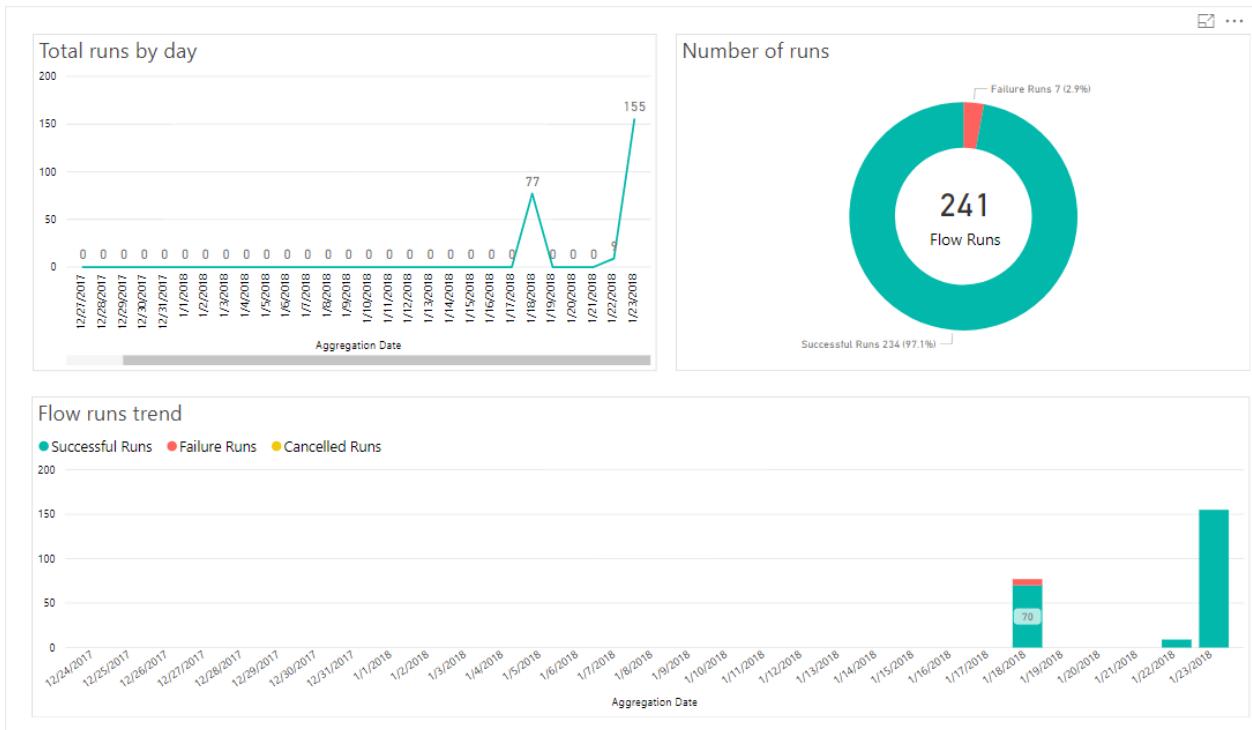
To send automated emails to different recipients, create separate flows. In these additional flows, change the recipient email address in the “To” field, and the email subject line in the “Subject” field. Creating new runbooks in Azure Automation with customized PowerShell scripts (such as with change 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:

The screenshot shows an Microsoft Outlook message window titled "Automatic tuning recommendations email notification - Message (Plain Text)". The message was sent by Danimir Ljepava at 7:29 PM. The message body contains the following text:

```
Name      : [REDACTED]
Account   : [REDACTED]
Environment : AzureCloud
Subscription : [REDACTED]
Tenant    : [REDACTED]
TokenCache : Microsoft.Azure.Commands.Common.Authentication.AuthenticationStoreTokenCache
VersionProfile :
ExtendedProperties : {}

SubscriptionId :
ResourceGroupName : [REDACTED]
ServerName       : [REDACTED]
DatabaseName     : [REDACTED]
Script          : DROP INDEX [REDACTED] ON [dbo].[test_hinted_drop_name]

SubscriptionId :
ResourceGroupName : [REDACTED]
ServerName       : [REDACTED]
DatabaseName     : [REDACTED]
Script          : CREATE NONCLUSTERED INDEX [REDACTED] ON
[CRM].[DataPoints] ([Name],[Money],[Power]) INCLUDE ([Hour], [System], [LastChanged]) WITH (ONLINE
= ON)
```

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](#).

# Manual tune query performance in Azure SQL Database

10/22/2018 • 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

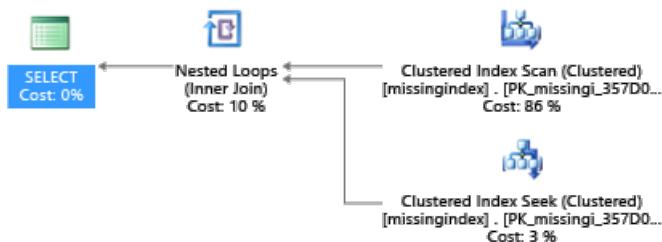
In this section, we look at some techniques that you can use to tune Azure SQL Database to gain the best performance for your application and run it at the lowest possible compute size. Some of these techniques match traditional SQL Server tuning best practices, but others are specific to Azure SQL Database. 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.

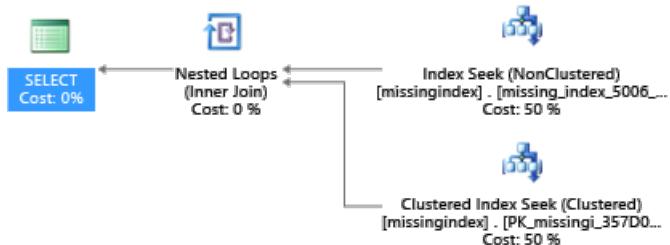
You can use this query to evaluate potential missing indexes:

```
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:

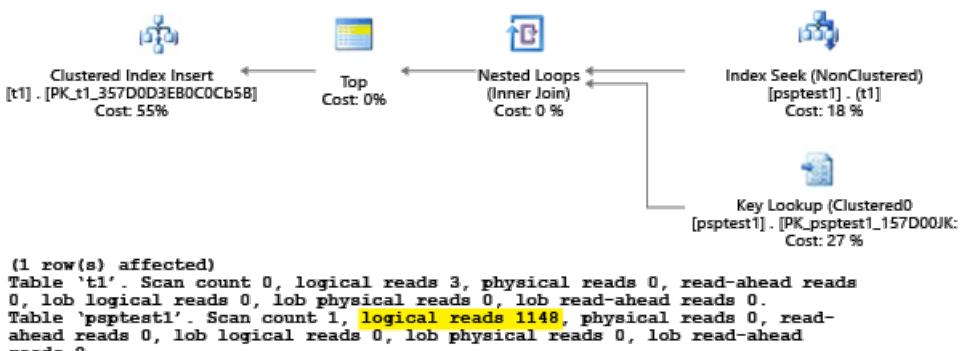


Because we executed the procedure by using the value 1, the resulting plan was optimal for the value 1 but was sub-optimal for all other values in the table. The result likely isn't what you would want if you were to pick each plan randomly, because the plan performs more slowly and uses more resources.

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 22:45:00.00 | 2013-06-18 22:50:00.00 | resource1     | Basic | 11.57                | 100.00          | 1.00                           | 6.00                  |
| 2 | 2013-06-18 22:40:00.00 | 2013-06-18 22:45:00.00 | resource1     | Basic | 11.57                | 2.00            | 0.50                           | 5.00                  |
| 3 | 2013-06-18 22:35:00.00 | 2013-06-18 22:40:00.00 | resource1     | Basic | 11.57                | 45.00           | 0.00                           | 5.00                  |
| 4 | 2013-06-18 22:30:00.00 | 2013-06-18 22:35:00.00 | resource1     | Basic | 11.57                | 0.00            | 0.00                           | 0.00                  |
| 5 | 2013-06-18 22:25:00.00 | 2013-06-18 22:30:00.00 | resource1     | Basic | 11.57                | 52.00           | 0.00                           | 5.00                  |
| 6 | 2013-06-18 22:20:00.00 | 2013-06-18 22:25:00.00 | 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 a single 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 a single 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 a single 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 a single 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 Redis Cache](#), 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 Azure SQL Database using dynamic management views

10/25/2018 • 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 / 1000.0) 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, 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 / 1000.00) 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
    FND
```

```

        ) 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 high a CPU 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.statement_start_offset / 2) + 1,
((CASE r.statement_end_offset
WHEN -1 THEN
    DATALENGTH(text)
ELSE
    r.statement_end_offset
END - r.statement_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;

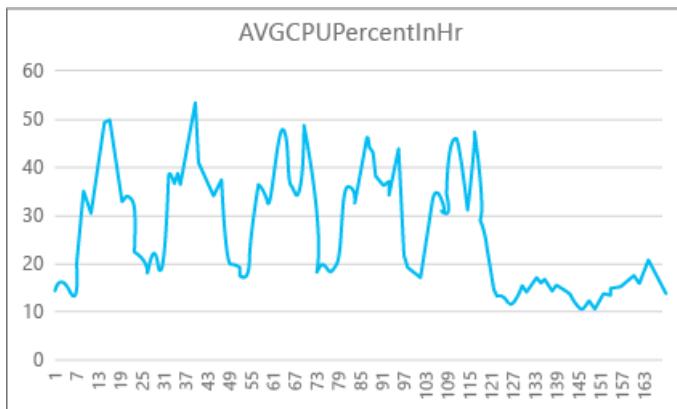
```

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 logical 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  | 2013-06-18 22:45:00.00 | 2013-06-18 22:50:00.00 | resource1     | Basic | 11.57                | 105.00          | 1.00                           | 6.00                  |
| 2  | 2013-06-18 22:40:00.00 | 2013-06-18 22:45:00.00 | resource1     | Basic | 11.57                | 2.00            | 0.50                           | 5.00                  |
| 3  | 2013-06-18 22:35:00.00 | 2013-06-18 22:40:00.00 | resource1     | Basic | 11.57                | 45.00           | 0.00                           | 5.00                  |
| 4  | 2013-06-18 22:30:00.00 | 2013-06-18 22:35:00.00 | resource1     | Basic | 11.57                | 0.00            | 0.00                           | 0.00                  |
| 5  | 2013-06-18 22:25:00.00 | 2013-06-18 22:30:00.00 | resource1     | Basic | 11.57                | 52.00           | 0.00                           | 5.00                  |
| 6  | 2013-06-18 22:20:00.00 | 2013-06-18 22:25:00.00 | resource1     | Basic | 11.57                | 1.00            | 17.00                          | 3.00                  |
| 7  | 2013-06-18 22:15:00.00 | 2013-06-18 22:20:00.00 | resource1     | Basic | 11.57                | 12.00           | 11.00                          | 23.00                 |
| 8  | 2013-06-18 22:10:00.00 | 2013-06-18 22:15:00.00 | resource1     | Basic | 11.57                | 75.00           | 69.00                          | 24.00                 |
| 9  | 2013-06-18 22:05:00.00 | 2013-06-18 22:10:00.00 | resource1     | Basic | 11.57                | 60.00           | 53.00                          | 54.00                 |
| 10 | 2013-06-18 22:00:00.00 | 2013-06-18 22:05:00.00 | resource1     | Basic | 11.57                | 12.00           | 18.00                          | 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:

1. 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;
```

2. 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());
```

3. 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

9/25/2018 • 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](#)

# Monitor In-Memory OLTP storage

9/25/2018 • 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

9/25/2018 • 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   |
|---|---|
| <b>sys.database_event_session_actions</b> | Returns a row for each action on each event of an event session.                            |
| <b>sys.database_event_session_events</b>  | Returns a row for each event in an event session.   |
| <b>sys.database_event_session_fields</b>  | Returns a row for each customize-able column that was explicitly set on events and targets. |
| <b>sys.database_event_session_targets</b> | Returns a row for each event target for an event session.                                   |
| <b>sys.database_event_sessions</b>        | 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  |
|--|--|
| <b>sys.dm_xe_database_session_event_actions</b>  | Returns information about event session actions.                             |
| <b>sys.dm_xe_database_session_events</b>         | Returns information about session events.                                    |
| <b>sys.dm_xe_database_session_object_columns</b> | Shows the configuration values for objects that are bound to a session.      |
| <b>sys.dm_xe_database_session_targets</b>        | Returns information about session targets.                                   |
| <b>sys.dm_xe_database_sessions</b>               | 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:
  - <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

9/25/2018 • 9 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

- 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-AzureStorageAccount**.

## Phase 1: PowerShell code for Azure Storage container

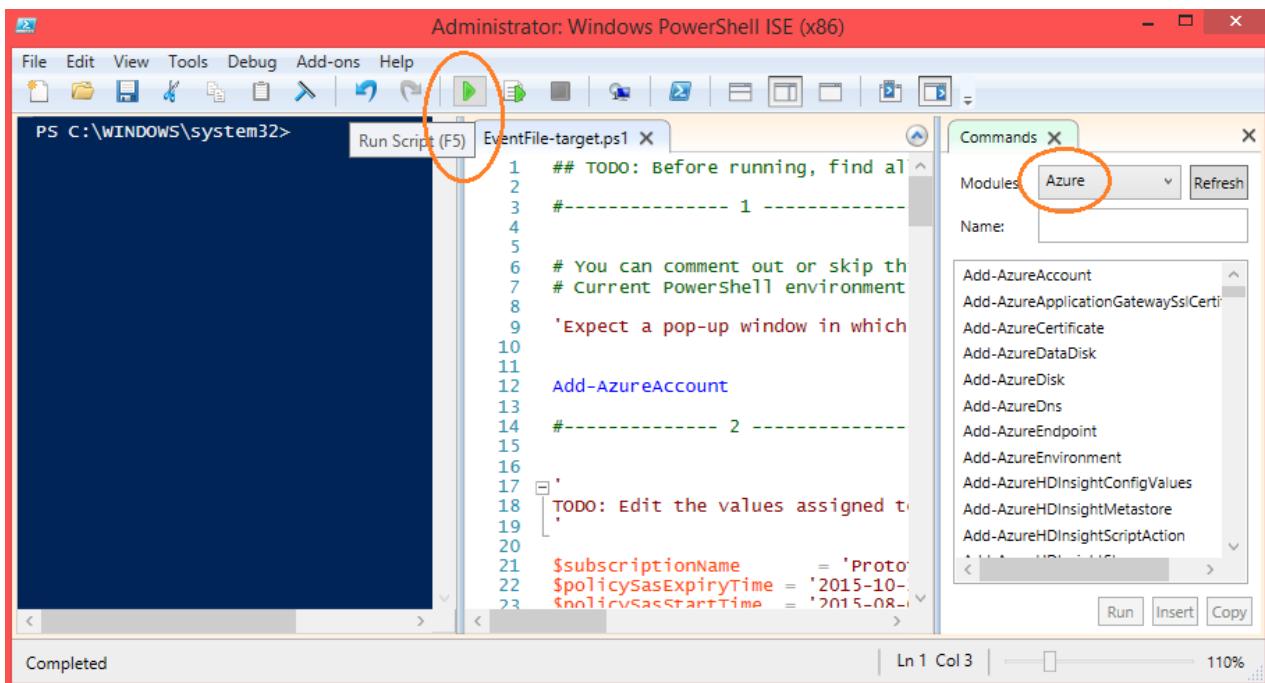
This PowerShell is phase 1 of the two-phase code sample.

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 run the cmdlet Import-Module for the AzureRm module. For reference documentation, see [PowerShell Module Browser](#).

```
## 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-AzureRmAccount (or Connect-AzureRmAccount), just one time.';
#Connect-AzureRmAccount; # Same as Connect-AzureRmAccount.

#----- 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 = 'West US';
$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-AzureRmSubscription -Subscription $subscriptionName;

#----- 4 -----


'Clean up the old Azure Storage Account after any previous run,
before continuing this new run.';

If ($storageAccountName)
{
    Remove-AzureRmStorageAccount `

        -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-AzureRmStorageAccount `

    -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-AzureRmStorageAccountKey `

        -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-AzureStorageContext `

    -StorageAccountName $storageAccountName `

    -StorageAccountKey  $accessKey_ForStorageAccount;

'Create a container within the storage account.
';

$containerObjectInStorageAccount = New-AzureStorageContainer `

    -Name      $containerName `

    -Context   $context;

```

```

'Create a security policy to be applied to the SAS token.
';

New-AzureStorageContainerStoredAccessPolicy ` 
    -Container $containerName ` 
    -Context $context ` 
    -Policy $policySasToken ` 
    -Permission $policySasPermission ` 
    -ExpiryTime $policySasExpiryTime ` 
    -StartTime $policySasStartTime;

'

Generate a SAS token for the container.
';
Try
{
    $sasTokenWithPolicy = New-AzureStorageContainerSASToken ` 
        -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-AzureRmStorageAccount - 
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

```

```

----- Then stop the session.

----- Note: If the target fails to attach,
----- the session must be stopped and restarted.

ALTER
EVENT SESSION
    gmeventsessionname240b
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
    gmeventsessionname240b
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.
    'https://gmstorageaccountxevent.blob.core.windows.net/gmcontainerxevent/anyfilenameelix242b',
    null, null, null
);
GO

----- Step 6. Clean up. -------

DROP
EVENT SESSION
    gmeventsessionname240b
ON DATABASE;
GO

DROP DATABASE SCOPED CREDENTIAL
-- TODO: Assign AzureStorageAccount name, and the associated Container name.
[https://gmstorageaccountxevent.blob.core.windows.net/gmcontainerxevent]
;
GO

DROP TABLE gmTabEmployee;
GO

PRINT 'Use PowerShell Remove-AzureStorageAccount to delete your Azure Storage account!';
GO

```

```
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 element is displayed which shows one UPDATE statement.

Here is one 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

9/25/2018 • 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_azuresqldb51')
BEGIN
    DROP EVENT SESSION eventsession_gm_azuresqldb51
        ON DATABASE;
END
GO

CREATE
EVENT SESSION eventsession_gm_azuresqldb51
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_azuresqldb51
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_azuresqlDb51'
    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_azuresqlDb51
    ON DATABASE
    STATE = STOP;
GO

```

---- Step set 7.

```

ALTER EVENT SESSION eventsession_gm_azuresqlDb51
    ON DATABASE
    DROP TARGET package0.ring_buffer;
GO

```

---- Step set 8.

```

DROP EVENT SESSION eventsession_gm_azuresqlDb51

```

```

DROP EVENT SESSION eventSessionOn_gm_azuresqldb
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 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;
</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](#)

# Use Azure portal to create alerts for Azure SQL Database and Data Warehouse

9/25/2018 • 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. This article also provides best practices for setting alert periods.

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

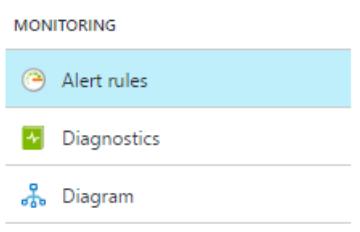
- [Azure portal](#)
- [PowerShell](#)
- [command-line interface \(CLI\)](#)
- [Azure Monitor REST API](#)

## 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. This step is different for SQL DB and elastic pools versus SQL DW:

- **SQL DB & Elastic pools ONLY:** Select **Alerts** or **Alert rules** under the MONITORING section. The text and icon may vary slightly for different resources.



- **SQL DW ONLY:** Select **Monitoring** under the COMMON TASKS section. Click the **DWU Usage** graph.

**COMMON TASKS**

- Load Data
- Scale
- Monitoring
- Open In Visual Studio

3. Select the **Add alert** command and fill in the fields.

**Add an alert rule**

\* Resource   
\*\*\*\*\*/SampleDB (Databases) (servers...)

\* Name   
MyNewDBAlert

Description   
Description

\* Metric   
CPU percentage

40% 30% 20% 10% 0%

5 PM Apr 10 6 AM 12 PM

\* Condition   
greater than

\* Threshold   
40 %

\* Period   
Over the last 5 minutes

Email owners, contributors, and readers

Additional administrator email(s)   
admin@contoso.com

Webhook   
http://www.contoso.com/dowork?param

Learn more about [configuring webhooks](#)

**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                |

| Resource Type      | Metric Name                | Friendly Name                           | Aggregation Type | Minimum Alert Time Window |
|--------------------|----------------------------|---|------------------|---------------------------|
| SQL database       | xtp_storage_percent        | In-Memory OLTP storage percent(Preview) | Average          | 5 minutes                 |
| 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 | storage                    | Total database size                     | Maximum          | 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                |

| RESOURCE TYPE      | METRIC NAME             | FRIENDLY NAME                | AGGREGATION TYPE | MINIMUM ALERT TIME WINDOW |
|--------------------|-------------------------|------------------------------|------------------|---------------------------|
| SQL data warehouse | service_level_objective | Service tier of the database | Total            | 10 minutes                |
| SQL data warehouse | dwu_limit               | dwu limit                    | Maximum          | 10 minutes                |
| 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.

# Troubleshoot connection issues to Azure SQL Database

10/2/2018 • 4 minutes to read • [Edit Online](#)

When the connection to Azure SQL Database fails, you receive [error messages](#). This article is a centralized topic that helps you troubleshoot Azure SQL Database connectivity issues. It introduces [the common causes](#) of connection issues, recommends [a troubleshooting tool](#) that helps you identify the problem, and provides troubleshooting steps to solve [transient errors](#) and [persistent or non-transient errors](#).

If you encounter the connection issues, try the troubleshoot steps that are described in this article.

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

## Cause

Connection problems may be caused by any of the following:

- Failure to apply best practices and design guidelines during the application design process. See [SQL Database Development Overview](#) to get started.
- Azure SQL Database reconfiguration
- Firewall settings
- Connection time-out
- Incorrect login information
- Maximum limit reached on some Azure SQL Database resources

Generally, connection issues to Azure SQL Database can be classified as follows:

- [Transient errors \(short-lived or intermittent\)](#)
- [Persistent or non-transient errors \(errors that regularly recur\)](#)

## Try the troubleshooter for Azure SQL Database connectivity issues

If you encounter a specific connection error, try [this tool](#), which will help you quickly identify and resolve your problem.

## Troubleshoot transient errors

When an application connects to an Azure SQL database, you receive the following error message:

```
Error code 40613: "Database <x> on server <y> is not currently available. Please retry the connection later. If the problem persists, contact customer support, and provide them the session tracing ID of <z>"
```

### NOTE

This error message is typically transient (short-lived).

This error occurs when the Azure database is being moved (or reconfigured) and your application loses its

connection to the SQL database. SQL 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.

### 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. Review the [Transient errors](#) section and the best practices and design guidelines at [SQL Database Development Overview](#) for more information and general retry strategies. Then, see code samples at [Connection Libraries for SQL Database and SQL Server](#) for specifics.
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.

## Troubleshoot persistent errors

If the application persistently fails to connect to Azure SQL Database, it usually indicates an issue with one of the following:

- Firewall configuration. The Azure SQL database or client-side firewall is blocking connections to Azure SQL Database.
- Network reconfiguration on the client side: for example, a new IP address or a proxy server.
- User error: for example, mistyped connection parameters, such as the server name in the connection string.

### Steps to resolve persistent connectivity issues

1. Set up [firewall rules](#) to allow the client IP address. For temporary testing purposes, set up a firewall rule using 0.0.0.0 as the starting IP address range and using 255.255.255.255 as the ending IP address range. This will open the server to all IP addresses. If this resolves your connectivity issue, remove this rule and create a firewall rule for an appropriately limited IP address or address range.
2. On all firewalls between the client and the Internet, make sure that port 1433 is open for outbound connections. Review [Configure the Windows Firewall to Allow SQL Server Access](#) and [Hybrid Identity Required Ports and Protocols](#) for additional pointers related to additional ports that you need to open for Azure Active Directory authentication.
3. Verify your connection string and other connection settings. See the Connection String section in the [connectivity issues topic](#).
4. Check service health in the dashboard. If you think there's a regional outage, see [Recover from an outage](#) for steps to recover to a new region.

## Next steps

- [Search the documentation on Microsoft Azure](#)
- [View the latest updates to the Azure SQL Database service](#)

## Additional resources

- [SQL Database Development Overview](#)

- General transient fault-handling guidance
- Connection libraries for SQL Database and SQL Server

# Managing Azure SQL databases using Azure Automation

10/8/2018 • 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 highly-reliable and highly-available workflow execution engine that scales to meet your needs as your organization grows. In Azure Automation, processes can be kicked off manually, by 3rd-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 3rd 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 3rd 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](#)

# Manage groups of databases with Elastic Database Jobs

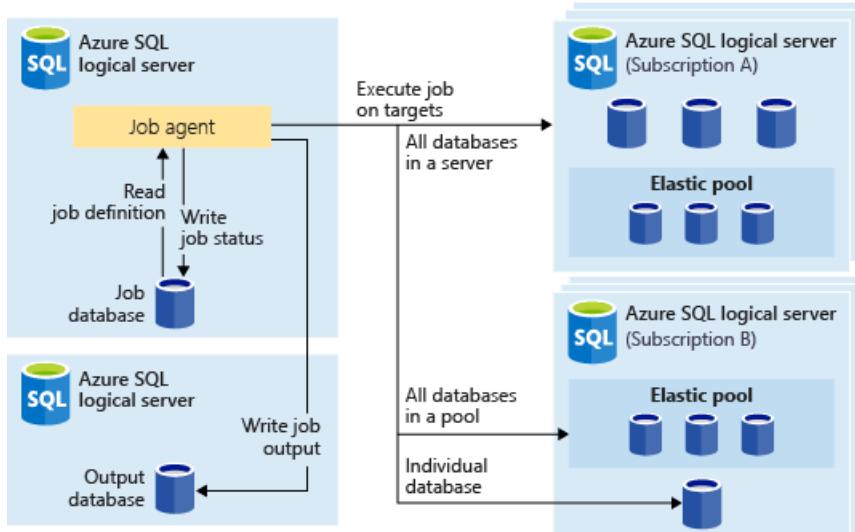
9/24/2018 • 11 minutes to read • [Edit Online](#)

**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:



## Why use elastic jobs?

### Manage many databases

- Schedule administrative tasks 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).
- 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.

### Reduce overhead

- Normally, you must connect to each database independently in order to run Transact-SQL statements or perform other administrative tasks. A job handles the task of logging in to each database in the target group.

You also define, maintain and persist Transact-SQL scripts to be executed across a group of Azure SQL databases.

## Accounting

- Jobs log the status of execution for each database. You also get automatic retry when failures occur.

## Flexibility

- Define custom groups of Azure SQL databases, and define schedules for running a job.

# Elastic Job components

| COMPONENT                | DESCRIPTION (ADDITIONAL DETAILS ARE BELOW THE TABLE)   |
|--------------------------|--|
| <b>Elastic Job agent</b> | The Azure resource you create to run and manage Jobs.  |
| <b>Job database</b>      | An Azure SQL database the job agent uses to store job related data, job definitions, etc.  |
| <b>Target group</b>      | The set of servers, pools, databases, and shard maps to run a job against.   |
| <b>Job</b>               | A job is a unit of work that is comprised 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 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 tier. The recommended service tier of the *Job database* is S1 or higher, but really depends on the performance needs of your job(s): number of job steps, how many times, and how frequently jobs are run. For example, an S0 database might be sufficient for a job agent that runs few jobs an hour, but running a job every minute might not be performant enough, and a higher service tier might be better.

## 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:

- **Azure SQL 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.

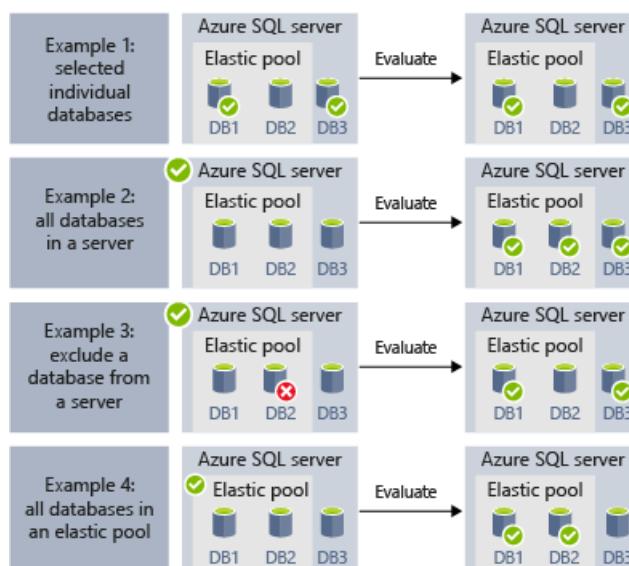
## TIP

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:

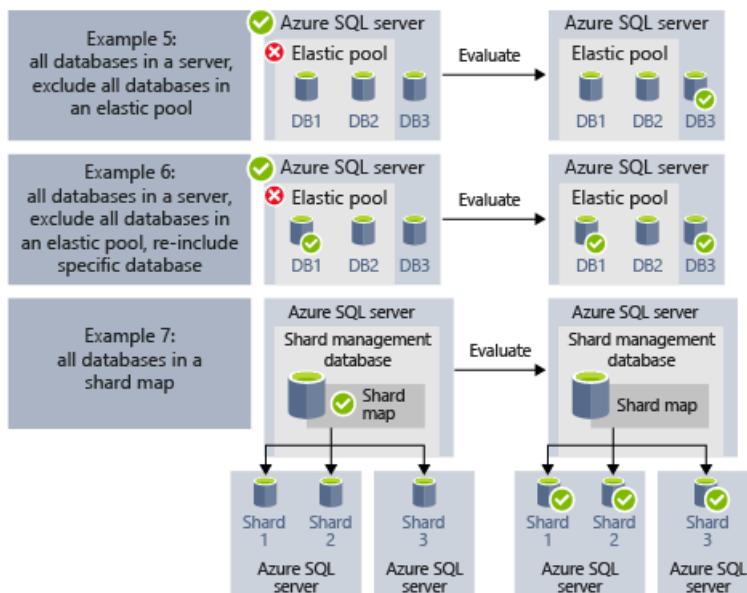


**Example 1** shows a target group that consists of a list of individual databases. When a job step is executed using this target group, the job step's action will be executed in each of those databases.

**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.

## 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*.

## Workflow to create, configure, and manage jobs

### Create and configure the agent

1. Create or identify an empty S0 or higher SQL database. This will be used as the *Job database* during Elastic Job agent creation.
2. Create an Elastic Job agent in the [portal](#), or with [PowerShell](#).

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.

[Learn more](#)

\* Name  
ConstosoAgent

\* Subscription  
ContosoSubscription

Preview terms  
Accepted

\* Job database  
[Configure required settings](#)

Pin to dashboard

**Create**   [Automation options](#)

Select server  
contososerver

Search to filter databases...

| DATABASE NAME | PRICING TIER         |
|---------------|----------------------|
| JobDatabase   | Standard S0: 10 D... |

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](#).

contosoagent  
Elastic Job agent - PREVIEW

Search (Ctrl+)

**Overview** (highlighted with a red box)

Activity log

Access control (IAM)

Tags

SETTINGS

Quick start

Jobs

Target groups

Credentials

Properties

Locks

Automation script

Refresh Delete

Resource group (change)  
ContosoRg

Status  
Ready

Location  
West US 2

Subscription (change)

Subscription ID

Elastic Jobs is currently in preview and you can see the list of currently executing jobs below. Click here to learn more about how to create and manage your jobs, target groups, and credentials.

Latest 100 job executions

| STATUS      | JOB NAME | JOB EXECUTION ID                     | START TIME             | END TIME               | DURATION |
|-------------|----------|--------------------------------------|------------------------|------------------------|----------|
| In Progress | Job1     | 05594811-cc30-4c38-bdc0-d5deact796bc | 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      |

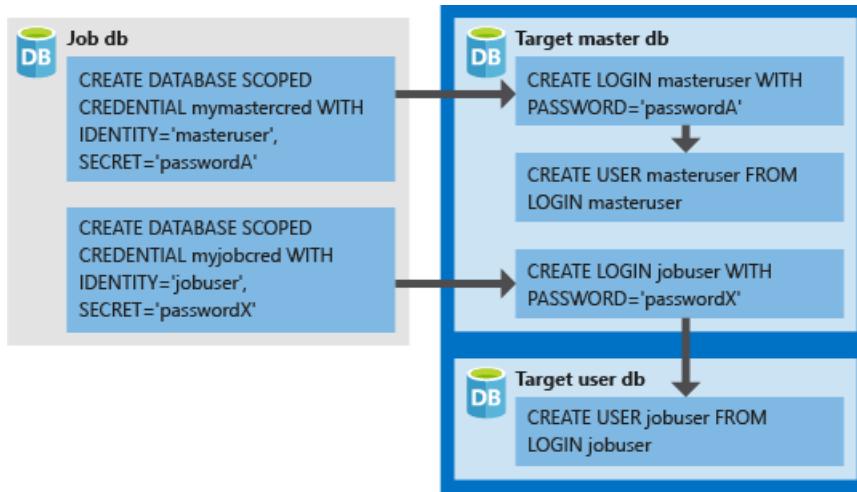
## Credentials for running jobs

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 are expected to 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 additional 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 which 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.

## Differences between Elastic Jobs and SQL Server Agent

It is worth noting a couple of differences between SQL Server Agent (available on-premises and as part of SQL Database Managed Instance), and the Azure SQL Database Elastic Job agent (now available for SQL Database and SQL Data Warehouse).

|                          | ELASTIC JOBS  | SQL SERVER AGENT  |
|--------------------------|---|---|
| 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 logical 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 single 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)                            |

## 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 EXIST (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 \(T-SQL\)](#)

# Create an Elastic Job agent using PowerShell

9/24/2018 • 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

If you don't have already have an Azure subscription, [create a free account](#) before you begin.

Install the **AzureRM.Sql** 4.8.1-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 AzureRM.Sql preview cmdlets side by side with existing AzureRM.Sql version
Install-Module -Name AzureRM.Sql -AllowPrerelease -RequiredVersion 4.8.1-preview -Force

# Import the AzureRM.Sql 4.8.1 module
Import-Module AzureRM.Sql -RequiredVersion 4.8.1

# Confirm if module successfully imported - if the imported version is 4.8.1, then continue
Get-Module AzureRM.Sql
```

## 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 script below also creates a second server with 2 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 your Azure account
Connect-AzureRmAccount

# Create a resource group
Write-Output "Creating a resource group..."
$ResourceGroupName = Read-Host "Please enter a resource group name"
$Location = Read-Host "Please enter an Azure Region"
$Rg = New-AzureRmResourceGroup -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-AzureRmSqlServer -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-AzureRmSqlServerFirewallRule -AllowAllAzureIPs
$AgentServer

# Create the job database
Write-Output "Creating a blank SQL database to be used as the Job Database..."
$JobDatabaseName = "JobDatabase"
$JobDatabase = New-AzureRmSqlDatabase -ResourceGroupName $ResourceGroupName -ServerName $AgentServerName -
DatabaseName $JobDatabaseName -RequestedServiceObjectiveName "S0"
$JobDatabase

```

```

# Create a target server and some sample databases - uses the same admin credential as the agent server just
for simplicity
Write-Output "Creating target server..."
$TargetServerName = Read-Host "Please enter a target server name"
$TargetServerName = $TargetServerName + "-" + [guid]::NewGuid()
$TargetServer = New-AzureRmSqlServer -ResourceGroupName $ResourceGroupName -Location $Location -ServerName
$TargetServerName -ServerVersion "12.0" -SqlAdministratorCredentials ($AdminCred)

# Set target server firewall rules to allow all Azure IPs
$TargetServer | New-AzureRmSqlServerFirewallRule -AllowAllAzureIPs
$TargetServer | New-AzureRmSqlServerFirewallRule -StartIpAddress 0.0.0.0 -EndIpAddress 255.255.255.255 -
FirewallRuleName AllowAll
$TargetServer

# Create some sample databases to execute jobs against...
$db1 = New-AzureRmSqlDatabase -ResourceGroupName $ResourceGroupName -ServerName $TargetServerName -
DatabaseName "TargetDb1"
$db1
$db2 = New-AzureRmSqlDatabase -ResourceGroupName $ResourceGroupName -ServerName $TargetServerName -
DatabaseName "TargetDb2"
$db2

```

## Enable the Elastic Jobs preview for your subscription

To use Elastic Jobs, register the feature in your Azure subscription by running the following command (this only needs to be run once in each subscription where you want to use Elastic Jobs):

```
Register-AzureRmProviderFeature -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-AzureRmSqlElasticJobAgent** cmdlet requires an Azure SQL database to already exist, so the *ResourceGroupName*, *ServerName*, and *DatabaseName* parameters must all point to existing resources.

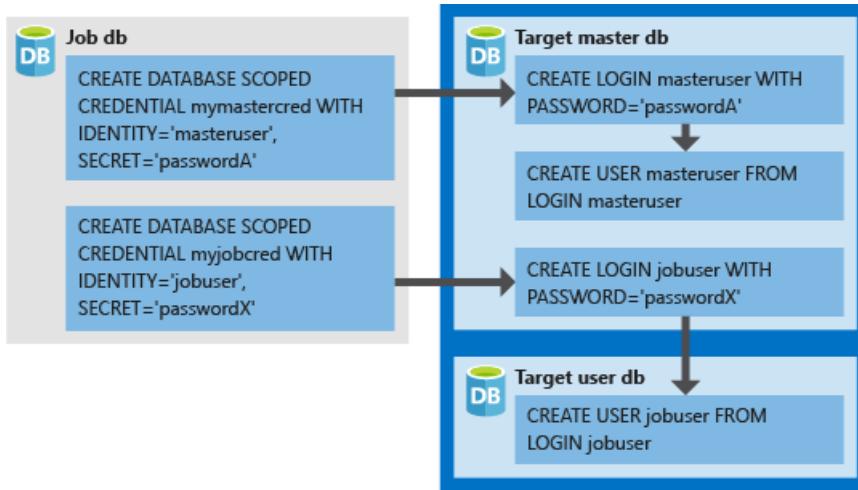
```
Write-Output "Creating job agent..."  
$AgentName = Read-Host "Please enter a name for your new Elastic Job agent"  
$JobAgent = $JobDatabase | New-AzureRmSqlElasticJobAgent -Name $AgentName  
$JobAgent
```

## Create job credentials so that jobs can execute scripts on its targets

Jobs use database scoped credentials to connect to the target databases specified by the target group upon execution. 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
# - Create the master user from master user login
# - Create the 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 of the target databases
# - Create the jobuser from jobuser login
# - Make sure they have the right permissions for successful 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-AzureRmSqlElasticJobCredential -Name "masteruser" -Credential $MasterCred

$JobCred = New-Object -TypeName "System.Management.Automation.PSCredential" -ArgumentList "jobuser",
$LoginPasswordSecure
$JobCred = $JobAgent | New-AzureRmSqlElasticJobCredential -Name "jobuser" -Credential $JobCred

```

## Define the target databases you want 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-AzureRmSqlElasticJobTargetGroup -Name 'ServerGroup'
$ServerGroup | Add-AzureRmSqlElasticJobTarget -ServerName $TargetServerName -RefreshCredentialName
$MasterCred.CredentialName

# Create ServerGroup with an exclusion of Db2
$ServerGroupExcludingDb2 = $JobAgent | New-AzureRmSqlElasticJobTargetGroup -Name 'ServerGroupExcludingDb2'
$ServerGroupExcludingDb2 | Add-AzureRmSqlElasticJobTarget -ServerName $TargetServerName -RefreshCredentialName
$MasterCred.CredentialName
$ServerGroupExcludingDb2 | Add-AzureRmSqlElasticJobTarget -ServerName $TargetServerName -Database
$Db2.DatabaseName -Exclude

```

## Create a job

```

Write-Output "Creating a new job"
$JobName = "Job1"
$Job = $JobAgent | New-AzureRmSqlElasticJob -Name $JobName -RunOnce
$Job

```

## Create a job step

This example defines 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 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-AzureRmSqlElasticJobStep -Name "step1" -TargetGroupName $ServerGroup.TargetGroupName -
CredentialName $JobCred.CredentialName -CommandText $SqlText1
$Job | Add-AzureRmSqlElasticJobStep -Name "step2" -TargetGroupName $ServerGroupExcludingDb2.TargetGroupName -
CredentialName $JobCred.CredentialName -CommandText $SqlText2

```

## Run the job

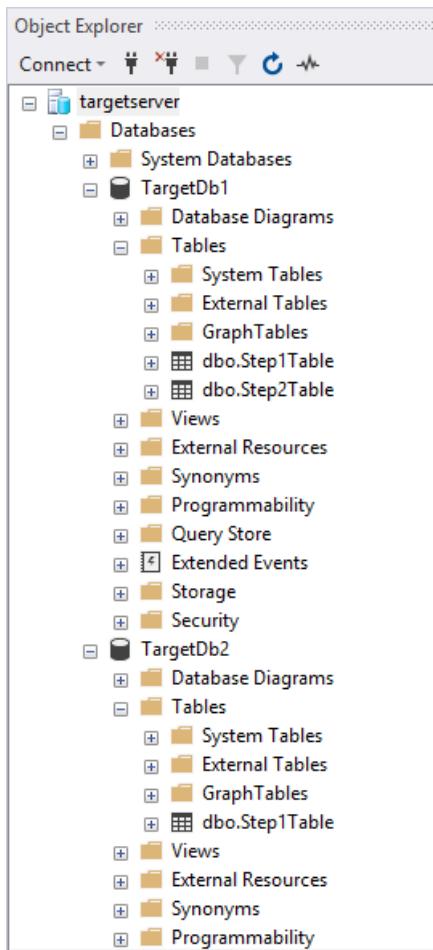
To start the job immediately, run the following command:

```

Write-Output "Start a new execution of the job..."
$JobExecution = $Job | Start-AzureRmSqlElasticJob
$JobExecution

```

After successful completion you should see two new tables in TargetDb1, and only one new table in TargetDb2:



## Monitor status of job executions

The following snippets get job execution details:

```
# Get the latest 10 executions run
$JobAgent | Get-AzureRmSqlElasticJobExecution -Count 10

# Get the job step execution details
$JobExecution | Get-AzureRmSqlElasticJobStepExecution

# Get the job target execution details
$JobExecution | Get-AzureRmSqlElasticJobTargetExecution -Count 2
```

## 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-AzureRmSqlElasticJob -IntervalType Hour -IntervalCount 1 -StartTime (Get-Date) -Enable
```

## 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, do not clean up the resources created in this article. If you do not plan to continue, use the following steps to delete all resources created in this article.

```
Remove-AzureRmResourceGroup -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

9/24/2018 • 39 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 a single database

The following example shows how to execute a job against all databases in a 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 look to create the table to store the returned results in. As a result the login associated with the credential used for the output credential will need to have sufficient permissions to perform this. 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\_Internal\_Execution\_ID" on the internal\_execution\_id column.

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 logical server. refresh\_credential\_name is nvarchar(128), with no default.

[ **@server\_name** = ] 'server\_name'

The name of the logical 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 databases within a server or Elastic pool at time of execution, when a logical 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="#">jobs_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.             |

### **jobs\_executions view**

[jobs].[jobs\_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 logical 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<br>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\_verions]

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\_groups\_members view**

[jobs].[target\_groups\_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 logical 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

10/30/2018 • 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 have already have an Azure subscription, [create a free account](#) before you begin.

Install the **AzureRM.Sql** 4.8.1-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 AzureRM.Sql preview cmdlets side by side with existing AzureRM.Sql version
Install-Module -Name AzureRM.Sql -AllowPrerelease -RequiredVersion 4.8.1-preview -Force

# Import the AzureRM.Sql 4.8.1 module
Import-Module AzureRM.Sql -RequiredVersion 4.8.1

# Confirm if module successfully imported - if the imported version is 4.8.1, then continue
Get-Module AzureRM.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-AzureRmProviderFeature -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-AzureRmSqlDatabase -ResourceGroupName <resourceGroupName> -ServerName <serverName> -DatabaseName <databaseName>

# Create a new elastic job agent
$agent = $db | New-AzureRmSqlElasticJobAgent -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-AzureRmSubscription -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-AzureRmSqlElasticJobCredential -ParentObject $agent -Name $oldCredName -Credential
$oldCredential
        }
        catch [System.Management.Automation.PSArgumentException]
        {
            $cred = Get-AzureRmSqlElasticJobCredential -ParentObject $agent -Name $oldCredName
            $cred = Set-AzureRmSqlElasticJobCredential -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
                }
            }
            $j++
        }
    }
}
```

```

        }

        # 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-AzureRmSqlElasticJobTargetGroup -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-AzureRmSqlElasticJobTarget -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-AzureRmSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -DatabaseName $databaseName
-Exclude
    }
    else {
        $t = Add-AzureRmSqlElasticJobTarget -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-AzureRmSqlElasticJobTarget -ParentObject $tg -ServerName $serverName -ShardMapName $smName -
        DatabaseName $databasename -RefreshCredentialName $credName -Exclude
    }
    else {
        $t = Add-AzureRmSqlElasticJobTarget -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-AzureRmSqlElasticJobTargetGroup -ParentObject $agent -Name $tgName
    return $tg
}
catch [System.Management.Automation.PSArgumentException] {
    $tg = Get-AzureRmSqlElasticJobTargetGroup -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-AzureRmSqlElasticJob -ParentObject $agent -Name $jobName ` 
        -Description $jobDescription -IntervalType $intervalType -IntervalCount $intervalCount ` 
        -StartTime $startTime -EndTime $endTime
    return $job
}
catch [System.Management.Automation.PSArgumentException] {
    $job = Get-AzureRmSqlElasticJob -ParentObject $agent -Name $jobName
    $job = $job | Set-AzureRmSqlElasticJob -Description $jobDescription -IntervalType $intervalType - 
    IntervalCount $intervalCount ` 
        -StartTime $startTime -EndTime $endTime
    return $job
}
}

# Create or update a job that runs once
else {
    try {
        $job = New-AzureRmSqlElasticJob -ParentObject $agent -Name $jobName ` 
            -Description $jobDescription -RunOnce
        return $job
    }
    catch [System.Management.Automation.PSArgumentException] {
        $job = Get-AzureRmSqlElasticJob -ParentObject $agent -Name $jobName
        $job = $job | Set-AzureRmSqlElasticJob -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-AzureRmSqlDatabase -ResourceGroupName $job.ResourceGroupName -ServerName
        $outputServerName -Databasename $outputDatabaseName

        try {
            $jobStep = $job | Add-AzureRmSqlElasticJobStep -Name $defaultJobStepName ` 
                -TargetGroupName $targetGroupName -CredentialName $credentialName -CommandText $commandText ` 
                -OutputDatabaseObject $outputDatabase ` 
                -OutputSchemaName $outputSchemaName -OutputTableName $outputTableName ` 
                -OutputCredentialName $outputCredentialName
        }
        catch [System.Management.Automation.PSArgumentException] {
            $jobStep = $job | Get-AzureRmSqlElasticJobStep -Name $defaultJobStepName
            $jobStep = $jobStep | Set-AzureRmSqlElasticJobStep -TargetGroupName $targetGroupName ` 
                -CredentialName $credentialName -CommandText $commandText ` 
                -OutputDatabaseObject $outputDatabase ` 
                -OutputSchemaName $outputSchemaName -OutputTableName $outputTableName ` 
                -OutputCredentialName $outputCredentialName
        }
    }
}
else {
}

```

```

try {
    $jobStep = $job | Add-AzureRmSqlElasticJobStep -Name $defaultJobStepName -TargetGroupName
$targetGroupName -CredentialName $credentialName -CommandText $commandText
}
catch [System.Management.Automation.PSArgumentException] {
    $jobStep = $job | Get-AzureRmSqlElasticJobStep -Name $defaultJobStepName
    $jobStep = $jobStep | Set-AzureRmSqlElasticJobStep -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-AzureRmSqlElasticJobCredential  
$targetGroups = $agent | Get-AzureRmSqlElasticJobTargetGroup  
$jobs = $agent | Get-AzureRmSqlElasticJob  
$steps = $jobs | Get-AzureRmSqlElasticJobStep
```

To test that jobs are executing correctly, start them:

```
$jobs | Start-AzureRmSqlElasticJob
```

For any jobs that were running on a schedule, remember to enable them so that they can run in the background:

```
$jobs | Set-AzureRmSqlElasticJob -Enable
```

## Next steps

- [Create and manage Elastic Jobs using PowerShell](#)
- [Create and manage Elastic Jobs using Transact-SQL \(T-SQL\)](#)

# Managing scaled-out cloud databases

10/30/2018 • 7 minutes to read • [Edit Online](#)

## IMPORTANT

This article is for the customer-hosted version of *Elastic Database jobs*. Elastic Database jobs are being deprecated and replaced with new Azure-hosted **Elastic Database jobs**. For new jobs, use the latest **Elastic Database jobs**. If you currently use the older customer-hosted jobs, see [Migrate to the new Elastic Database jobs](#) for directions and migration scripts to quickly upgrade to the latest version.

**Elastic Database jobs** is a customer-hosted Azure Cloud Service that enables the execution of ad-hoc and scheduled administrative tasks, which are called **jobs**. With jobs, you can easily and reliably manage large groups of Azure SQL databases by running Transact-SQL scripts to perform administrative operations.

To manage scaled-out sharded databases, the **Elastic Database jobs** feature (preview) enables you to reliably execute a Transact-SQL (T-SQL) script across a group of databases, including:

- a custom-defined collection of databases (explained below)
- all databases in an [elastic pool](#)
- a shard set (created using [Elastic Database client library](#)).

For more information about the concept of a sharded database, see [Sharding a SQL Server Database](#).

## Documentation

- [Install the Elastic Database job components](#).
- [Get started with Elastic Database jobs](#).
- [Create and manage jobs using PowerShell](#).
- [Create and manage scaled out Azure SQL databases](#)



## Why use jobs

## Manage

Easily do schema changes, credentials management, reference data updates, performance data collection or tenant (customer) telemetry collection.

## Reports

Aggregate data from a collection of Azure SQL databases into a single destination table.

## Reduce overhead

Normally, you must connect to each database independently in order to run Transact-SQL statements or perform other administrative tasks. A job handles the task of logging in to each database in the target group. You also define, maintain and persist Transact-SQL scripts to be executed across a group of Azure SQL databases.

## Accounting

Jobs run the script and log the status of execution for each database. You also get automatic retry when failures occur.

## Flexibility

Define custom groups of Azure SQL databases, and define schedules for running a job.

### NOTE

In the Azure portal, only a reduced set of functions limited to SQL Azure elastic pools is available. Use the PowerShell APIs to access the full set of current functionality.

## Applications

- Perform administrative tasks, such as deploying a new schema.
- Update reference data-product information common across all databases. Or schedules automatic updates every weekday, after hours.
- Rebuild indexes to improve query performance. The rebuilding can be configured 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.
- 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.

## Elastic Database jobs: end-to-end

1. Install the **Elastic Database jobs** components. For more information, see [Installing Elastic Database jobs](#). If the installation fails, see [how to uninstall](#).
2. Use the PowerShell APIs to access more functionality, for example creating custom-defined database collections, adding schedules and/or gathering results sets. Use the portal for simple installation and creation/monitoring of jobs limited to execution against a **elastic pool**.
3. Create encrypted credentials for job execution and [add the user \(or role\) to each database in the group](#).
4. Create an idempotent T-SQL script that can be run against every database in the group.
5. Follow these steps to create jobs using the Azure portal: [Creating and managing Elastic Database jobs](#).
6. Or use PowerShell scripts: [Create and manage a SQL Database elastic database jobs using PowerShell \(preview\)](#).

## Idempotent scripts

The scripts must be **idempotent**. In simple terms, "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 has been successfully run twice.

A simple tactic is to test for the existence of an object before creating it.

```
IF NOT EXIST (some_object)
-- Create the object
```

Similarly, a script must be able to execute successfully by logically testing for and countering any conditions it finds.

## Failures and logs

If a script fails after multiple attempts, the job logs the error and continues. After a job ends (meaning a run against all databases in the group), you can check its list of failed attempts. The logs provide details to debug faulty scripts.

## Group types and creation

There are two kinds of groups:

1. Shard sets
2. Custom groups

Shard set groups are created using the [Elastic Database tools](#). When you create a shard set group, databases are added or removed from the group automatically. For example, a new shard will be automatically in the group when you add it to the shard map. A job can then be run against the group.

Custom groups, on the other hand, are rigidly defined. You must explicitly add or remove databases from custom groups. If a database in the group is dropped, the job will attempt to run the script against the database resulting in an eventual failure. Groups created using the Azure portal currently are custom groups.

## Components and pricing

The following components work together to create an Azure Cloud service that enables ad-hoc execution of administrative jobs. The components are installed and configured automatically during setup, in your subscription. You can identify the services as they all have the same auto-generated name. The name is unique, and consists of the prefix "edj" followed by 21 randomly generated characters.

- Azure Cloud Service

Elastic database jobs (preview) is delivered as a customer-hosted Azure Cloud service to perform execution of the requested tasks. From the portal, the service is deployed and hosted in your Microsoft Azure subscription. The default deployed service runs with the minimum of two worker roles for high availability. The default size of each worker role (ElasticDatabaseJobWorker) runs on an A0 instance. For pricing, see [Cloud services pricing](#).

- Azure SQL Database

The service uses an Azure SQL Database known as the **control database** to store all of the job metadata. The default service tier is a S0. For pricing, see [SQL Database Pricing](#).

- Azure Service Bus

An Azure Service Bus is for coordination of the work within the Azure Cloud Service. See [Service Bus](#)

## Pricing.

- Azure Storage

An Azure Storage account is used to store diagnostic output logging in the event that an issue requires further debugging (see [Enabling Diagnostics in Azure Cloud Services and Virtual Machines](#)). For pricing, see [Azure Storage Pricing](#).

## How Elastic Database jobs work

1. An Azure SQL Database is designated a **control database** which stores all meta-data and state data.
2. The control database is accessed by the **job service** to launch and track jobs to execute.
3. Two different roles communicate with the control database:
  - Controller: Determines which jobs require tasks to perform the requested job, and retries failed jobs by creating new job tasks.
  - Job Task Execution: Carries out the job tasks.

### Job task types

There are multiple types of job tasks that carry out execution of jobs:

- ShardMapRefresh

Queries the shard map to determine all the databases used as shards
- ScriptSplit

Splits the script across 'GO' statements into batches
- ExpandJob

Creates child jobs for each database from a job that targets a group of databases
- ScriptExecution

Executes a script against a particular database using defined credentials
- Dacpac

Applies a DACPAC to a particular database using particular credentials

## End-to-end job execution work-flow

1. Using either the Portal or the PowerShell API, a job is inserted into the **control database**. The job requests execution of a Transact-SQL script against a group of databases using specific credentials.
2. The controller identifies the new job. Job tasks are created and executed to split the script and to refresh the group's databases. Lastly, a new job is created and executed to expand the job and create new child jobs where each child job is specified to execute the Transact-SQL script against an individual database in the group.
3. The controller identifies the created child jobs. For each job, the controller creates and triggers a job task to execute the script against a database.
4. After all job tasks have completed, the controller updates the jobs to a completed state. At any point during job execution, the PowerShell API can be used to view the current state of job execution. All times returned by the PowerShell APIs are represented in UTC. If desired, a cancellation request can be initiated to stop a job.

## Next steps

Install the components, then [create](#) and add a log in to each database in the group of databases. To further understand job creation and management, see [creating and managing elastic database jobs](#). See also [Getting started with 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](#).

# Installing Elastic Database jobs overview

10/30/2018 • 6 minutes to read • [Edit Online](#)

## IMPORTANT

This article is for the customer-hosted version of *Elastic Database jobs*. Elastic Database jobs are being deprecated and replaced with new Azure-hosted **Elastic Database jobs**. For new jobs, use [the latest Elastic Database jobs](#). If you currently use the older customer-hosted jobs, see [Migrate to the new Elastic Database jobs](#) for directions and migration scripts to quickly upgrade to the latest version.

**Elastic Database jobs** can be installed via PowerShell or through the Azure portal. You can gain access to create and manage jobs using the PowerShell API only if you install the PowerShell package. Additionally, the PowerShell APIs provide significantly more functionality than the portal at this point in time.

If you have already installed **Elastic Database jobs** through the Portal from an existing **elastic pool**, the latest Powershell preview includes scripts to upgrade your existing installation. It is highly recommended to upgrade your installation to the latest **Elastic Database jobs** components in order to take advantage of new functionality exposed via the PowerShell APIs.

## Prerequisites

- An Azure subscription. For a free trial, see [Free trial](#).
- Azure PowerShell. Install the latest version using the [Web Platform Installer](#). For detailed information, see [How to install and configure Azure PowerShell](#).
- NuGet Command-line Utility is used to install the Elastic Database jobs package. For more information, see <http://docs.nuget.org/docs/start-here/installing-nuget>.

## Download and import the Elastic Database jobs PowerShell package

1. Launch Microsoft Azure PowerShell command window and navigate to the directory where you downloaded NuGet Command-line Utility (nuget.exe).
2. Download and import **Elastic Database jobs** package into the current directory with the following command:

```
PS C:\>.\nuget install Microsoft.Azure.SqlDatabase.Jobs -prerelease
```

The **Elastic Database jobs** files are placed in the local directory in a folder named **Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x** where x.x.xxxx.x reflects the version number. The PowerShell cmdlets (including required client .dlls) are located in the **tools\ElasticDatabaseJobs** sub-directory and the PowerShell scripts to install, upgrade and uninstall also reside in the **tools** sub-directory.

3. Navigate to the tools sub-directory under the Microsoft.Azure.SqlDatabase.Jobs.x.x.xxx.x folder by typing cd tools, for example:

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*>cd tools
```

4. Execute the `.\InstallElasticDatabaseJobsCmdlets.ps1` script to copy the `ElasticDatabaseJobs` directory into `$home\Documents\WindowsPowerShell\Modules`. This will also automatically import the module for use,

for example:

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*\tools>Unblock-File  
.\\InstallElasticDatabaseJobsCmdlets.ps1  
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*\tools>.\InstallElasticDatabaseJobsCmdlets.ps1
```

## Install the Elastic Database jobs components using PowerShell

1. Launch a Microsoft Azure PowerShell command window and navigate to the \tools sub-directory under the Microsoft.Azure.SqlDatabase.Jobs.x.x.xxx.x folder: Type cd \tools

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*>cd tools
```

2. Execute the .\InstallElasticDatabaseJobs.ps1 PowerShell script and supply values for its requested variables. This script will create the components described in [Elastic Database jobs components and pricing](#) along with configuring the Azure Cloud Service to appropriately use the dependent components.

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*\tools>Unblock-File  
.\\InstallElasticDatabaseJobs.ps1  
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*\tools>.\InstallElasticDatabaseJobs.ps1
```

When you run this command a window opens asking for a **user name** and **password**. This is not your Azure credentials, enter the user name and password that will be the administrator credentials you want to create for the new server.

The parameters provided on this sample invocation can be modified for your desired settings. The following provides more information on the behavior of each parameter:

| PARAMETER             | DESCRIPTION   |
|-----------------------|---|
| ResourceGroupName     | Provides the Azure resource group name created to contain the newly created Azure components. This parameter defaults to “_ElasticDatabaseJob”. It is not recommended to change this value.   |
| ResourceGroupLocation | Provides the Azure location to be used for the newly created Azure components. This parameter defaults to the Central US location.  |
| ServiceWorkerCount    | Provides the number of service workers to install. This parameter defaults to 1. A higher number of workers can be used to scale out the service and to provide high availability. It is recommended to use “2” for deployments that require high availability of the service.  |
| ServiceVmSize         | Provides the VM size for usage within the Cloud Service. This parameter defaults to A0. Parameters values of A0/A1/A2/A3 are accepted which cause the worker role to use an ExtraSmall/Small/Medium/Large size, respectively. For more information on worker role sizes, see <a href="#">Elastic Database jobs components and pricing</a> . |

|                                |  |
|--------------------------------|--|
| SqlServerDatabaseSlo           | Provides the compute size for a Standard edition. This parameter defaults to S0. Parameter values of S0/S1/S2/S3/S4/S6/S9/S12 are accepted which cause the Azure SQL Database to use the respective compute size. For more information on SQL Database compute sizes, see <a href="#">Elastic Database jobs components and pricing</a> . |
| SqlServerAdministratorUserName | Provides the admin user name for the newly created Azure SQL Database server. When not specified, a PowerShell credentials window will open to prompt for the credentials.   |
| SqlServerAdministratorPassword | Provides the admin password for the newly created Azure SQL Database server. When not provided, a PowerShell credentials window will open to prompt for the credentials.   |

For systems that target having large numbers of jobs running in parallel against a large number of databases, it is recommended to specify parameters such as: -ServiceWorkerCount 2 -ServiceVmSize A2 -SqlServerDatabaseSlo S2.

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.dll.x.x.xxx.*\tools>Unblock-File .\InstallElasticDatabaseJobs.ps1
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.dll.x.x.xxx.*\tools>.\InstallElasticDatabaseJobs.ps1 -
    ServiceWorkerCount 2 -ServiceVmSize A2 -SqlServerDatabaseSlo S2
```

## Update an existing Elastic Database jobs components installation using PowerShell

**Elastic Database jobs** can be updated within an existing installation for scale and high-availability. This process allows for future upgrades of service code without having to drop and recreate the control database. This process can also be used within the same version to modify the service VM size or the server worker count.

To update the VM size of an installation, run the following script with parameters updated to the values of your choice.

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.dll.x.x.xxx.*\tools>Unblock-File .\UpdateElasticDatabaseJobs.ps1
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.dll.x.x.xxx.*\tools>.\UpdateElasticDatabaseJobs.ps1 -ServiceVmSize A1
    -ServiceWorkerCount 2
```

| PARAMETER          | DESCRIPTION  |
|--------------------|--|
| ResourceGroupName  | Identifies the Azure resource group name used when the Elastic Database job components were initially installed. This parameter defaults to "__ElasticDatabaseJob". Since it is not recommended to change this value, you shouldn't have to specify this parameter.            |
| ServiceWorkerCount | Provides the number of service workers to install. This parameter defaults to 1. A higher number of workers can be used to scale out the service and to provide high availability. It is recommended to use "2" for deployments that require high availability of the service. |

## ServiceVmSize

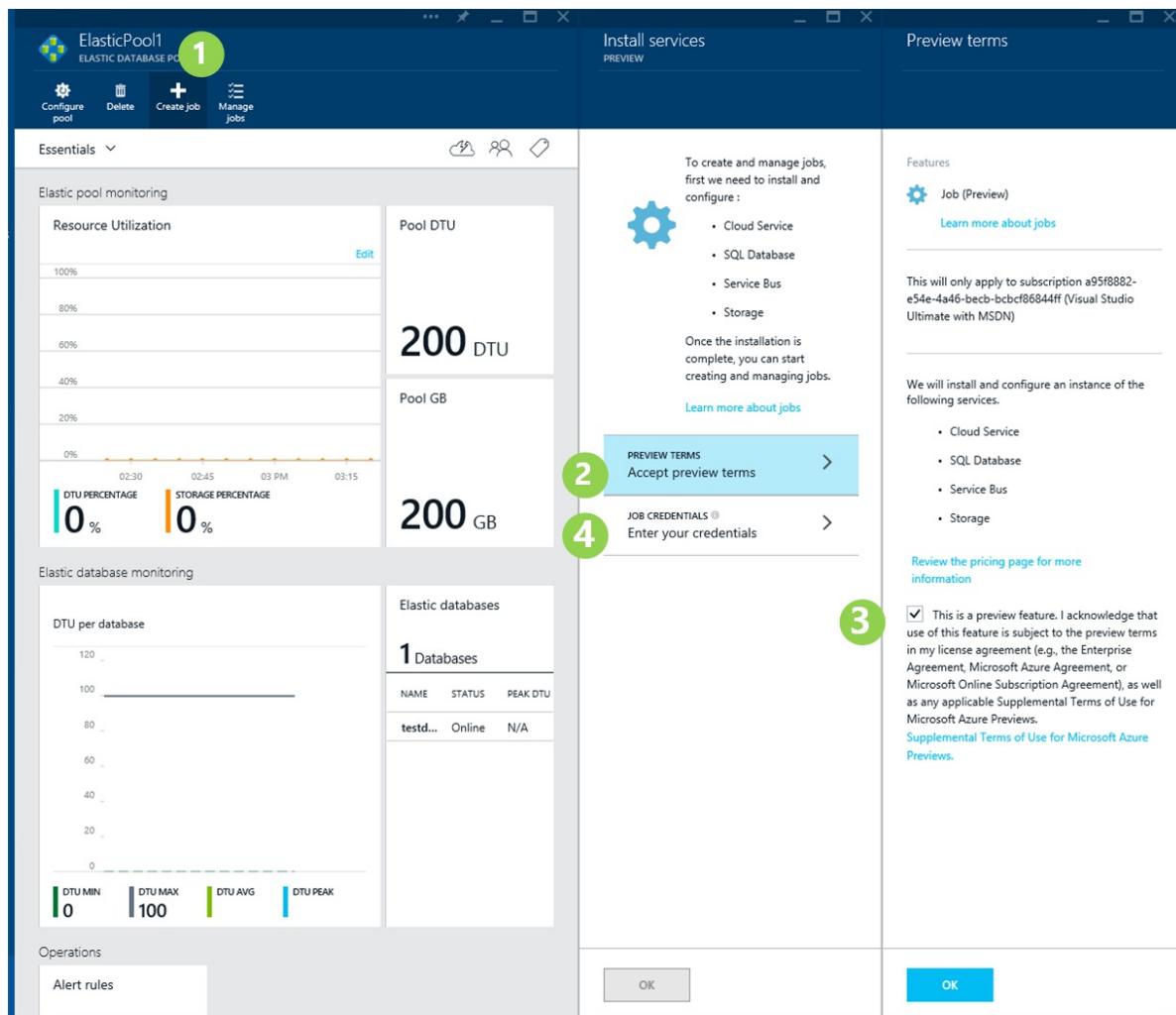
Provides the VM size for usage within the Cloud Service. This parameter defaults to A0. Parameters values of A0/A1/A2/A3 are accepted which cause the worker role to use an ExtraSmall/Small/Medium/Large size, respectively. For more information on worker role sizes, see [Elastic Database jobs components and pricing](#).

## Install the Elastic Database jobs components using the Portal

Once you have [created an elastic pool](#), you can install **Elastic Database jobs** components to enable execution of administrative tasks against each database in the elastic pool. Unlike when using the **Elastic Database jobs** PowerShell APIs, the portal interface is currently restricted to only executing against an existing pool.

**Estimated time to complete:** 10 minutes.

1. From the dashboard view of the elastic pool via the [Azure portal](#), click **Create job**.
2. If you are creating a job for the first time, you must install **Elastic Database jobs** by clicking **PREVIEW TERMS**.
3. Accept the terms by clicking the checkbox.
4. In the "Install services" view, click **JOB CREDENTIALS**.



5. Type a user name and password for a database admin. As part of the installation, a new Azure SQL Database server is created. Within this new server, a new database, known as the control database, is created and used to contain the meta data for Elastic Database jobs. The user name and password created here are used for the purpose of logging in to the control database. A separate credential is used for script execution against the databases within the pool.



- Click the OK button. The components are created for you in a few minutes in a new [Resource group](#). The new resource group is pinned to the start board, as shown below. Once created, elastic database jobs (Cloud Service, SQL Database, Service Bus, and Storage) are all created in the group.

A screenshot of the Microsoft Azure start board. On the left is a vertical navigation bar with icons for HOME, NOTIFICATIONS (1), BROWSE, ACTIVE, BILLING, and HELP. The main area is titled "Microsoft Azure PREVIEW". It features several pinned resources:
 

- ElasticPool1 ELASTIC POOL**: Status "Ready".
- testdb-ddove SQL DATABASE**: Status "Online".
- Provisioning Elastic database job**: This card is highlighted with a red border.

 Other visible cards include "Browse everything" (Access all your resources, all in one place), "Service health MY RESOURCES" (world map with green dots), "Billing VISUAL STUDIO ULTIMATE WITH MSDN" (CREDIT LEFT: 150.00 USD, DAYS LEFT: 27), "Marketplace" (with icons for Docker, Cloudera, and Azure), "Help + support" (with a blue icon), "Feedback" (with a heart icon), "Portal settings" (with a gear icon), "Classic portal" (with a monitor icon), and "What's new" (with an info icon).

- If you attempt to create or manage a job while elastic database jobs is installing, when providing **Credentials** you will see the following message.

ElasticPool1  
ELASTIC DATABASE POOL

Configure pool Delete Create job Manage jobs

Essentials ^

Resource group **Group**

Status **Ready**

Location **East US 2**

Subscription name **Visual Studio Ultimate with MSDN**

Subscription ID

Server name

Pricing tier **Standard**

Elastic database pool settings **200 DTU, 200 GB**

Elastic databases **1 Databases**

Elastic database settings **0-100 DTU**

All settings →

Elastic pool monitoring

Resource Utilization

100%  
80%  
60%  
40%  
20%  
0%  
03 PM 03:15 03:30 03:45

DTU PERCENTAGE STORAGE PERCENTAGE

Pool DTU **200 DTU**

Pool GB **200 GB**

OK

Job credentials

Elastic database job deployment is in progress.

Database admin login **cloudSA**

Password **\*\*\*\*\***

If uninstallation is required, delete the resource group. See [How to uninstall the Elastic Database job components](#).

## Next steps

Ensure a credential with the appropriate rights for script execution is created on each database in the group, for more information see [Securing your SQL Database](#). See [Creating and managing an Elastic Database jobs](#) to get started.

# Create and manage scaled out Azure SQL databases using elastic jobs (preview)

10/30/2018 • 3 minutes to read • [Edit Online](#)

## IMPORTANT

This article is for the customer-hosted version of *Elastic Database jobs*. Elastic Database jobs are being deprecated and replaced with new Azure-hosted **Elastic Database jobs**. For new jobs, use [the latest Elastic Database jobs](#). If you currently use the older customer-hosted jobs, see [Migrate to the new Elastic Database jobs](#) for directions and migration scripts to quickly upgrade to the latest version.

**Elastic Database jobs** simplify management of groups of databases by executing administrative operations such as schema changes, credentials management, reference data updates, performance data collection or tenant (customer) telemetry collection. Elastic Database jobs is currently available through the Azure portal and PowerShell cmdlets. However, the Azure portal surfaces reduced functionality limited to execution across all databases in an [elastic pool](#). To access additional features and execution of scripts across a group of databases including a custom-defined collection or a shard set (created using [Elastic Database client library](#)), see [Creating and managing jobs using PowerShell](#). For more information about jobs, see [Elastic Database jobs overview](#).

## Prerequisites

- An Azure subscription. For a free trial, see [Free trial](#).
- An elastic pool. See [About elastic pools](#).
- Installation of elastic database job service components. See [Installing the elastic database job service](#).

## Creating jobs

1. Using the [Azure portal](#), from an existing elastic database job pool, click **Create job**.
2. Type in the username and password of the database administrator (created at installation of Jobs) for the jobs control database (metadata storage for jobs).

The screenshot shows the 'Job credentials' configuration step in the 'Create job' blade. A green circle labeled '2' highlights the 'Database admin login' field, which contains 'cloudSA'. Below it is the 'Password' field, which contains a masked password. The background shows the main dashboard with resource utilization and database monitoring data.

3. In the **Create Job** blade, type a name for the job.
4. Type the user name and password to connect to the target databases with sufficient permissions for script execution to succeed.
5. Paste or type in the T-SQL script.
6. Click **Save** and then click **Run**.

The screenshot shows the 'Create job' blade with a T-SQL script pasted into the editor area. A green circle labeled '6' points to the 'Save' button at the top right of the editor. Other numbered circles (3, 4, 5) point to the 'Job name' field ('RLS-Policy'), the 'Username' field ('cloudSA'), and the script content respectively. The background shows the main dashboard with resource utilization and database monitoring data.

```

1 IF NOT EXISTS(SELECT * FROM SYS.SCHEMAS WHERE NAME = 'rls')
2 BEGIN
3     exec sp_executesql N'CREATE SCHEMA rls'
4 END
5 GO
6 IF NOT EXISTS(SELECT * FROM sys.objects WHERE type IN ('FN', 'IF'))
7 BEGIN
8     exec sp_executesql N'
9         CREATE FUNCTION rls.fn_franchiseAccessPredicate(@FranchiseOwnerUserName NVARCHAR(128))
10        RETURNS TABLE
11        WITH SCHEMABINDING
12        AS
13        RETURN
14        SELECT 1 AS fn_accessResult
15        FROM dbo.FranchiseAccess
16        WHERE USER_NAME() = FranchiseOwnerUserName AND @FranchiseOwnerUserName <= FranchiseOwnerUserName
17    END
18    GO
19 IF NOT EXISTS(SELECT * FROM sys.security_policies WHERE name = 'rls')
20 BEGIN
21     exec sp_executesql N'
22         CREATE SECURITY POLICY rls.franchiseAccessPolicy
23         ADD FILTER PREDICATE rls.fn_franchiseAccessPredicate(FranchiseOwnerUserName)
24     END
25 GO

```

## Run idempotent jobs

When you run a script against a set of databases, you must be sure that the script is idempotent. That is, the script must be able to run multiple times, even if it has failed before in an incomplete state. For example, when a script fails, the job will be automatically retried until it succeeds (within limits, as the retry logic will eventually cease the retrying). The way to do this is to use the an "IF EXISTS" clause and delete any found instance before creating a new object. An example is shown here:

```
IF EXISTS (SELECT name FROM sys.indexes  
          WHERE name = N'IX_ProductVendor_VendorID')  
DROP INDEX IX_ProductVendor_VendorID ON Purchasing.ProductVendor;  
GO  
CREATE INDEX IX_ProductVendor_VendorID  
ON Purchasing.ProductVendor (VendorID);
```

Alternatively, use an "IF NOT EXISTS" clause before creating a new instance:

```
IF NOT EXISTS (SELECT name FROM sys.tables WHERE name = 'TestTable')  
BEGIN  
CREATE TABLE TestTable(  
    TestTableId INT PRIMARY KEY IDENTITY,  
    InsertionTime DATETIME2  
);  
END  
GO  
  
INSERT INTO TestTable(InsertionTime) VALUES (sysutcdatetime());  
GO
```

This script then updates the table created previously.

```
IF NOT EXISTS (SELECT columns.name FROM sys.columns INNER JOIN sys.tables ON columns.object_id =  
tables.object_id WHERE tables.name = 'TestTable' AND columns.name = 'AdditionalInformation')  
BEGIN  
  
ALTER TABLE TestTable  
  
ADD AdditionalInformation NVARCHAR(400);  
END  
GO  
  
INSERT INTO TestTable(InsertionTime, AdditionalInformation) VALUES (sysutcdatetime(), 'test');  
GO
```

## Checking job status

After a job has begun, you can check on its progress.

1. From the elastic pool page, click **Manage jobs**.

The screenshot shows the 'franchisepool ELASTIC DATABASE POOL' interface. At the top, there are four main buttons: 'Configure pool', 'Delete', 'Create job', and 'Manage jobs'. The 'Manage jobs' button is highlighted with a red box. Below the buttons, there's a navigation bar with 'HOME', 'NOTIFICATIONS', and 'BROWSE' options. The main content area is titled 'Essentials' and includes sections for 'Elastic pool monitoring' (Resource Utilization chart) and 'Pool DTU'. A blue 'Edit' button is visible on the Resource Utilization chart.

2. Click on the name (a) of a job. The **STATUS** can be "Completed" or "Failed." The job's details appear (b) with its date and time of creation and running. The list (c) below the that shows the progress of the script against each database in the pool, giving its date and time details.

The screenshot shows the 'Manage jobs' window. On the left, a table lists two jobs: 'UpdateStats' (Completed) and 'RebuildIndex' (Completed). A green circle with 'a' highlights the 'UpdateStats' row. On the right, a detailed view of the 'UpdateStats' job is shown, with a green circle 'b' highlighting the 'Completed' status and execution details. Below this, a green circle 'c' highlights a table showing the progress of the 'UpdateStats' job across five databases: RecifeDB, CaracasDB, KatmanduDB, BeijingDB, and JohannesburgDB, all marked as 'Completed'.

| NAME         | STATUS    | START TIME           | END TIME             |
|--------------|-----------|----------------------|----------------------|
| UpdateStats  | Completed | 4/27/2015 2:48:00 AM | 4/27/2015 2:48:40 AM |
| RebuildIndex | Completed | 4/27/2015 2:50:31 AM | 4/27/2015 2:50:45 AM |

| Job execution details |                      |            |                      |
|-----------------------|----------------------|------------|----------------------|
| CREATED TIME          | 4/27/2015 2:47:41 AM | START TIME | 4/27/2015 2:48:00 AM |
| END TIME              | 4/27/2015 2:48:40 AM | STATUS     | Succeeded            |

| STATUS    | NAME           | START TIME           | END TIME             |
|-----------|----------------|----------------------|----------------------|
| Completed | RecifeDB       | 4/27/2015 2:48:09 AM | 4/27/2015 2:48:36 AM |
| Completed | CaracasDB      | 4/27/2015 2:48:09 AM | 4/27/2015 2:48:36 AM |
| Completed | KatmanduDB     | 4/27/2015 2:48:09 AM | 4/27/2015 2:48:38 AM |
| Completed | BeijingDB      | 4/27/2015 2:48:09 AM | 4/27/2015 2:48:36 AM |
| Completed | JohannesburgDB | 4/27/2015 2:48:09 AM | 4/27/2015 2:48:36 AM |

## Checking failed jobs

If a job fails, a log of its execution can be found. Click the name of the failed job to see its details.

| Name         | Status    | Start Time         | End Time          |
|--------------|-----------|--------------------|-------------------|
| CreateColumn | Retrying  | 3/9/2015, 9:28 PM  | --                |
| RLS-Script   | Running   | 3/9/2015, 2:05 AM  | --                |
| CreateIndex  | Running   | 3/9/2015, 4:05 AM  | --                |
| InsertRegion | Completed | 3/9/2015, 9:22 PM  | 3/9/2015, 9:26 AM |
| CreateTable  | Saved     | --                 | --                |
| CreateTable  | Failed    | 3/9/2015, 10:52 PM | --                |

## 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](#).

# Create and manage SQL Database elastic jobs using PowerShell (preview)

10/30/2018 • 18 minutes to read • [Edit Online](#)

## IMPORTANT

This article is for the customer-hosted version of *Elastic Database jobs*. Elastic Database jobs are being deprecated and replaced with new Azure-hosted **Elastic Database jobs**. For new jobs, use [the latest Elastic Database jobs](#). If you currently use the older customer-hosted jobs, see [Migrate to the new Elastic Database jobs](#) for directions and migration scripts to quickly upgrade to the latest version.

The PowerShell APIs for **Elastic Database jobs** (in preview), let you define a group of databases against which scripts will execute. This article shows how to create and manage **Elastic Database jobs** using PowerShell cmdlets. See [Elastic jobs overview](#).

## Prerequisites

- An Azure subscription. For a free trial, see [Free one-month trial](#).
- A set of databases created with the Elastic Database tools. See [Get started with Elastic Database tools](#).
- Azure PowerShell. For detailed information, see [How to install and configure Azure PowerShell](#).
- **Elastic Database jobs** PowerShell package: See [Installing Elastic Database jobs](#)

### Select your Azure subscription

To select the subscription you need your subscription Id (**-SubscriptionId**) or subscription name (**-SubscriptionName**). If you have multiple subscriptions you can run the **Get-AzureRmSubscription** cmdlet and copy the desired subscription information from the result set. Once you have your subscription information, run the following commandlet to set this subscription as the default, namely the target for creating and managing jobs:

```
Select-AzureRmSubscription -SubscriptionId {SubscriptionID}
```

The [PowerShell ISE](#) is recommended for usage to develop and execute PowerShell scripts against the Elastic Database jobs.

## Elastic Database jobs objects

The following table lists out all the object types of **Elastic Database jobs** along with its description and relevant PowerShell APIs.

| OBJECT TYPE | DESCRIPTION | RELATED POWERSHELL APIS |
|-------------|-------------|-------------------------|
|-------------|-------------|-------------------------|

|                                |   |  |
|--------------------------------|---|--|
| Credential                     | <p>Username and password to use when connecting to databases for execution of scripts or application of DACPACs.</p> <p>The password is encrypted before sending to and storing in the Elastic Database Jobs database. The password is decrypted by the Elastic Database Jobs service via the credential created and uploaded from the installation script.</p> | Get-AzureSqlJobCredential<br>New-AzureSqlJobCredential<br>Set-AzureSqlJobCredential                                      |
| Script                         | Transact-SQL script to be used for execution across databases. The script should be authored to be idempotent since the service will retry execution of the script upon failures.   | Get-AzureSqlJobContent<br>Get-AzureSqlJobContentDefinition<br>New-AzureSqlJobContent<br>Set-AzureSqlJobContentDefinition |
| DACPAC                         | Data-tier application package to be applied across databases.   | Get-AzureSqlJobContent<br>New-AzureSqlJobContent<br>Set-AzureSqlJobContentDefinition                                     |
| Database Target                | Database and server name pointing to an Azure SQL Database.   | Get-AzureSqlJobTarget<br>New-AzureSqlJobTarget   |
| Shard Map Target               | Combination of a database target and a credential to be used to determine information stored within an Elastic Database shard map.  | Get-AzureSqlJobTarget<br>New-AzureSqlJobTarget<br>Set-AzureSqlJobTarget  |
| Custom Collection Target       | Defined group of databases to collectively use for execution.   | Get-AzureSqlJobTarget<br>New-AzureSqlJobTarget   |
| Custom Collection Child Target | Database target that is referenced from a custom collection.  | Add-AzureSqlJobChildTarget<br>Remove-AzureSqlJobChildTarget  |
| Job                            | Definition of parameters for a job that can be used to trigger execution or to fulfill a schedule.  | Get-AzureSqlJob<br>New-AzureSqlJob<br>Set-AzureSqlJob  |
| Job Execution                  | Container of tasks necessary to fulfill either executing a script or applying a DACPAC to a target using credentials for database connections with failures handled in accordance to an execution policy.   | Get-AzureSqlJobExecution<br>Start-AzureSqlJobExecution<br>Stop-AzureSqlJobExecution<br>Wait-AzureSqlJobExecution         |

|                      |  |  |
|----------------------|--|--|
| Job Task Execution   | <p>Single unit of work to fulfill a job.</p> <p>If a job task is not able to successfully execute, the resulting exception message will be logged and a new matching job task will be created and executed in accordance to the specified execution policy.</p>                  | Get-AzureSqlJobExecution<br>Start-AzureSqlJobExecution<br>Stop-AzureSqlJobExecution<br>Wait-AzureSqlJobExecution |
| Job Execution Policy | <p>Controls job execution timeouts, retry limits and intervals between retries.</p> <p>Elastic Database jobs includes a default job execution policy which cause essentially infinite retries of job task failures with exponential backoff of intervals between each retry.</p> | Get-AzureSqlJobExecutionPolicy<br>New-AzureSqlJobExecutionPolicy<br>Set-AzureSqlJobExecutionPolicy               |
| Schedule             | <p>Time based specification for execution to take place either on a reoccurring interval or at a single time.</p>  | Get-AzureSqlJobSchedule<br>New-AzureSqlJobSchedule<br>Set-AzureSqlJobSchedule                                    |
| Job Triggers         | A mapping between a job and a schedule to trigger job execution according to the schedule.   | New-AzureSqlJobTrigger<br>Remove-AzureSqlJobTrigger  |

## Supported Elastic Database jobs group types

The job executes Transact-SQL (T-SQL) scripts or application of DACPACs across a group of databases. When a job is submitted to be executed across a group of databases, the job "expands" into child jobs where each performs the requested execution against a single database in the group.

There are two types of groups that you can create:

- **Shard Map** group: When a job is submitted to target a shard map, the job queries the shard map to determine its current set of shards, and then creates child jobs for each shard in the shard map.
- Custom Collection group: A custom defined set of databases. When a job targets a custom collection, it creates child jobs for each database currently in the custom collection.

## To set the Elastic Database jobs connection

A connection needs to be set to the jobs *control database* prior to using the jobs APIs. Running this cmdlet triggers a credential window to pop up requesting the user name and password created when installing Elastic Database jobs. All examples provided within this topic assume that this first step has already been performed.

Open a connection to the Elastic Database jobs:

```
Use-AzureSqlJobConnection -CurrentAzureSubscription
```

## Encrypted credentials within the Elastic Database jobs

Database credentials can be inserted into the jobs *control database* with its password encrypted. It is necessary to store credentials to enable jobs to be executed at a later time, (using job schedules).

Encryption works through a certificate created as part of the installation script. The installation script creates and uploads the certificate into the Azure Cloud Service for decryption of the stored encrypted passwords. The Azure Cloud Service later stores the public key within the jobs *control database* which enables the PowerShell API or Azure portal interface to encrypt a provided password without requiring the certificate to be locally installed.

The credential passwords are encrypted and secure from users with read-only access to Elastic Database jobs objects. But it is possible for a malicious user with read-write access to Elastic Database Jobs objects to extract a password. Credentials are designed to be reused across job executions. Credentials are passed to target databases when establishing connections. There are currently no restrictions on the target databases used for each credential, malicious user could add a database target for a database under the malicious user's control. The user could subsequently start a job targeting this database to gain the credential's password.

Security best practices for Elastic Database jobs include:

- Limit usage of the APIs to trusted individuals.
- Credentials should have the least privileges necessary to perform the job task. More information can be seen within this [Authorization and Permissions](#) SQL Server MSDN article.

### To create an encrypted credential for job execution across databases

To create a new encrypted credential, the **Get-Credential cmdlet** prompts for a user name and password that can be passed to the **New-AzureSqlJobCredential cmdlet**.

```
$credentialName = "{Credential Name}"
$databaseCredential = Get-Credential
$credential = New-AzureSqlJobCredential -Credential $databaseCredential -CredentialName $credentialName
Write-Output $credential
```

### To update credentials

When passwords change, use the **Set-AzureSqlJobCredential cmdlet** and set the **CredentialName** parameter.

```
$credentialName = "{Credential Name}"
Set-AzureSqlJobCredential -CredentialName $credentialName -Credential $credential
```

## To define an Elastic Database shard map target

To execute a job against all databases in a shard set (created using [Elastic Database client library](#)), use a shard map as the database target. This example requires a sharded application created using the Elastic Database client library. See [Getting started with Elastic Database tools sample](#).

The shard map manager database must be set as a database target and then the specific shard map must be specified as a target.

```
$shardMapCredentialName = "{Credential Name}"
$shardMapDatabaseName = "{ShardMapDatabaseName}" #example: ElasticScaleStarterKit_ShardMapManagerDb
$shardMapDatabaseServerName = "{ShardMapServerName}"
$shardMapName = "{MyShardMap}" #example: CustomerIDShardMap
$shardMapDatabaseTarget = New-AzureSqlJobTarget -DatabaseName $shardMapDatabaseName -ServerName
$shardMapDatabaseServerName
$shardMapTarget = New-AzureSqlJobTarget -ShardMapManagerCredentialName $shardMapCredentialName -
ShardMapManagerDatabaseName $shardMapDatabaseName -ShardMapManagerServerName $shardMapDatabaseServerName -
ShardMapName $shardMapName
Write-Output $shardMapTarget
```

# Create a T-SQL Script for execution across databases

When creating T-SQL scripts for execution, it is highly recommended to build them to be [idempotent](#) and resilient against failures. Elastic Database jobs will retry execution of a script whenever execution encounters a failure, regardless of the classification of the failure.

Use the [New-AzureSqlJobContent cmdlet](#) to create and save a script for execution and set the **-ContentName** and **-CommandText** parameters.

```
$scriptName = "Create a TestTable"

$scriptCommandText = "
IF NOT EXISTS (SELECT name FROM sys.tables WHERE name = 'TestTable')
BEGIN
    CREATE TABLE TestTable(
        TestTableId INT PRIMARY KEY IDENTITY,
        InsertionTime DATETIME2
    );
END
GO
INSERT INTO TestTable(InsertionTime) VALUES (sysutcdatetime());
GO"

$script = New-AzureSqlJobContent -ContentName $scriptName -CommandText $scriptCommandText
Write-Output $script
```

## Create a new script from a file

If the T-SQL script is defined within a file, use this to import the script:

```
$scriptName = "My Script Imported from a File"
$scriptPath = "{Path to SQL File}"
$scriptCommandText = Get-Content -Path $scriptPath
$script = New-AzureSqlJobContent -ContentName $scriptName -CommandText $scriptCommandText
Write-Output $script
```

## To update a T-SQL script for execution across databases

This PowerShell script updates the T-SQL command text for an existing script.

Set the following variables to reflect the desired script definition to be set:

```

$scriptName = "Create a TestTable"
$scriptUpdateComment = "Adding AdditionalInformation column to TestTable"
$scriptCommandText =
IF NOT EXISTS (SELECT name FROM sys.tables WHERE name = 'TestTable')
BEGIN
CREATE TABLE TestTable(
    TestTableId INT PRIMARY KEY IDENTITY,
    InsertionTime DATETIME2
);
END
GO

IF NOT EXISTS (SELECT columns.name FROM sys.columns INNER JOIN sys.tables ON columns.object_id =
tables.object_id WHERE tables.name = 'TestTable' AND columns.name = 'AdditionalInformation')
BEGIN
ALTER TABLE TestTable
ADD AdditionalInformation NVARCHAR(400);
END
GO

INSERT INTO TestTable(InsertionTime, AdditionalInformation) VALUES (sysutcdatetime(), 'test');
GO"

```

## To update the definition to an existing script

```

Set-AzureSqlJobContentDefinition -ContentName $scriptName -CommandText $scriptCommandText -Comment
$scriptUpdateComment

```

## To create a job to execute a script across a shard map

This PowerShell script starts a job for execution of a script across each shard in an Elastic Scale shard map.

Set the following variables to reflect the desired script and target:

```

$jobName = "{Job Name}"
$scriptName = "{Script Name}"
$shardMapServerName = "{Shard Map Server Name}"
$shardMapDatabaseName = "{Shard Map Database Name}"
$shardMapName = "{Shard Map Name}"
$credentialName = "{Credential Name}"
$shardMapTarget = Get-AzureSqlJobTarget -ShardManagerDatabaseName $shardMapDatabaseName -
ShardManagerServerName $shardMapServerName -ShardMapName $shardMapName
$job = New-AzureSqlJob -ContentName $scriptName -CredentialName $credentialName -JobName $jobName -TargetId
$shardMapTarget.TargetId
Write-Output $job

```

## To execute a job

This PowerShell script executes an existing job:

Update the following variable to reflect the desired job name to have executed:

```

$jobName = "{Job Name}"
$jobExecution = Start-AzureSqlJobExecution -JobName $jobName
Write-Output $jobExecution

```

## To retrieve the state of a single job execution

Use the **Get-AzureSqlJobExecution cmdlet** and set the **JobExecutionId** parameter to view the state of job execution.

```
$jobExecutionId = "{Job Execution Id}"
$jobExecution = Get-AzureSqlJobExecution -JobExecutionId $jobExecutionId
Write-Output $jobExecution
```

Use the same **Get-AzureSqlJobExecution cmdlet** with the **IncludeChildren** parameter to view the state of child job executions, namely the specific state for each job execution against each database targeted by the job.

```
$jobExecutionId = "{Job Execution Id}"
$jobExecutions = Get-AzureSqlJobExecution -JobExecutionId $jobExecutionId -IncludeChildren
Write-Output $jobExecutions
```

## To view the state across multiple job executions

The **Get-AzureSqlJobExecution cmdlet** has multiple optional parameters that can be used to display multiple job executions, filtered through the provided parameters. The following demonstrates some of the possible ways to use Get-AzureSqlJobExecution:

Retrieve all active top level job executions:

```
Get-AzureSqlJobExecution
```

Retrieve all top level job executions, including inactive job executions:

```
Get-AzureSqlJobExecution -IncludeInactive
```

Retrieve all child job executions of a provided job execution ID, including inactive job executions:

```
$parentJobExecutionId = "{Job Execution Id}"
Get-AzureSqlJobExecution -AzureSqlJobExecution -JobExecutionId $parentJobExecutionId -IncludeInactive -
IncludeChildren
```

Retrieve all job executions created using a schedule / job combination, including inactive jobs:

```
$jobName = "{Job Name}"
$scheduleName = "{Schedule Name}"
Get-AzureSqlJobExecution -JobName $jobName -ScheduleName $scheduleName -IncludeInactive
```

Retrieve all jobs targeting a specified shard map, including inactive jobs:

```
$shardMapServerName = "{Shard Map Server Name}"
$shardMapDatabaseName = "{Shard Map Database Name}"
$shardMapName = "{Shard Map Name}"
$target = Get-AzureSqlJobTarget -ShardMapManagerDatabaseName $shardMapDatabaseName -ShardMapManagerServerName
$shardMapServerName -ShardMapName $shardMapName
Get-AzureSqlJobExecution -TargetId $target.TargetId -IncludeInactive
```

Retrieve all jobs targeting a specified custom collection, including inactive jobs:

```
$customCollectionName = "{Custom Collection Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
Get-AzureSqlJobExecution -TargetId $target.TargetId -IncludeInactive
```

Retrieve the list of job task executions within a specific job execution:

```
$jobExecutionId = "{Job Execution Id}"
$jobTaskExecutions = Get-AzureSqlJobTaskExecution -JobExecutionId $jobExecutionId
Write-Output $jobTaskExecutions
```

Retrieve job task execution details:

The following PowerShell script can be used to view the details of a job task execution, which is particularly useful when debugging execution failures.

```
$jobTaskExecutionId = "{Job Task Execution Id}"
$jobTaskExecution = Get-AzureSqlJobTaskExecution -JobTaskExecutionId $jobTaskExecutionId
Write-Output $jobTaskExecution
```

## To retrieve failures within job task executions

The **JobTaskExecution object** includes a property for the lifecycle of the task along with a message property. If a job task execution failed, the lifecycle property will be set to *Failed* and the message property will be set to the resulting exception message and its stack. If a job did not succeed, it is important to view the details of job tasks that did not succeed for a given job.

```
$jobExecutionId = "{Job Execution Id}"
$jobTaskExecutions = Get-AzureSqlJobTaskExecution -JobExecutionId $jobExecutionId
Foreach($jobTaskExecution in $jobTaskExecutions)
{
    if($jobTaskExecution.Lifecycle -ne 'Succeeded')
    {
        Write-Output $jobTaskExecution
    }
}
```

## To wait for a job execution to complete

The following PowerShell script can be used to wait for a job task to complete:

```
$jobExecutionId = "{Job Execution Id}"
Wait-AzureSqlJobExecution -JobExecutionId $jobExecutionId
```

## Create a custom execution policy

Elastic Database jobs supports creating custom execution policies that can be applied when starting jobs.

Execution policies currently allow for defining:

- Name: Identifier for the execution policy.
- Job Timeout: Total time before a job will be canceled by Elastic Database Jobs.
- Initial Retry Interval: Interval to wait before first retry.
- Maximum Retry Interval: Cap of retry intervals to use.

- Retry Interval Backoff Coefficient: Coefficient used to calculate the next interval between retries. The following formula is used: (Initial Retry Interval) \* Math.pow((Interval Backoff Coefficient), (Number of Retries) - 2).
- Maximum Attempts: The maximum number of retry attempts to perform within a job.

The default execution policy uses the following values:

- Name: Default execution policy
- Job Timeout: 1 week
- Initial Retry Interval: 100 milliseconds
- Maximum Retry Interval: 30 minutes
- Retry Interval Coefficient: 2
- Maximum Attempts: 2,147,483,647

Create the desired execution policy:

```
$executionPolicyName = "{Execution Policy Name}"
$initialRetryInterval = New-TimeSpan -Seconds 10
$jobTimeout = New-TimeSpan -Minutes 30
$maxAttempts = 999999
$maximumRetryInterval = New-TimeSpan -Minutes 1
$retryIntervalBackoffCoefficient = 1.5
$executionPolicy = New-AzureSqlJobExecutionPolicy -ExecutionPolicyName $executionPolicyName -
InitialRetryInterval $initialRetryInterval -JobTimeout $jobTimeout -MaximumAttempts $maxAttempts -
MaximumRetryInterval $maximumRetryInterval
-RetryIntervalBackoffCoefficient $retryIntervalBackoffCoefficient
Write-Output $executionPolicy
```

## Update a custom execution policy

Update the desired execution policy to update:

```
$executionPolicyName = "{Execution Policy Name}"
$initialRetryInterval = New-TimeSpan -Seconds 15
$jobTimeout = New-TimeSpan -Minutes 30
$maxAttempts = 999999
$maximumRetryInterval = New-TimeSpan -Minutes 1
$retryIntervalBackoffCoefficient = 1.5
$updatedExecutionPolicy = Set-AzureSqlJobExecutionPolicy -ExecutionPolicyName $executionPolicyName -
InitialRetryInterval $initialRetryInterval -JobTimeout $jobTimeout -MaximumAttempts $maxAttempts -
MaximumRetryInterval $maximumRetryInterval -RetryIntervalBackoffCoefficient $retryIntervalBackoffCoefficient
Write-Output $updatedExecutionPolicy
```

## Cancel a job

Elastic Database Jobs supports cancellation requests of jobs. If Elastic Database Jobs detects a cancellation request for a job currently being executed, it will attempt to stop the job.

There are two different ways that Elastic Database Jobs can perform a cancellation:

1. Cancel currently executing tasks: If a cancellation is detected while a task is currently running, a cancellation will be attempted within the currently executing aspect of the task. For example: If there is a long running query currently being performed when a cancellation is attempted, there will be an attempt to cancel the query.
2. Canceling task retries: If a cancellation is detected by the control thread before a task is launched for execution, the control thread will avoid launching the task and declare the request as canceled.

If a job cancellation is requested for a parent job, the cancellation request will be honored for the parent job and for all of its child jobs.

To submit a cancellation request, use the [Stop-AzureSqlJobExecution cmdlet](#) and set the **JobExecutionId** parameter.

```
$jobExecutionId = "{Job Execution Id}"
Stop-AzureSqlJobExecution -JobExecutionId $jobExecutionId
```

## To delete a job and job history asynchronously

Elastic Database jobs supports asynchronous deletion of jobs. A job can be marked for deletion and the system will delete the job and all its job history after all job executions have completed for the job. The system will not automatically cancel active job executions.

Invoke [Stop-AzureSqlJobExecution](#) to cancel active job executions.

To trigger job deletion, use the [Remove-AzureSqlJob cmdlet](#) and set the **JobName** parameter.

```
$jobName = "{Job Name}"
Remove-AzureSqlJob -JobName $jobName
```

## To create a custom database target

You can define custom database targets either for direct execution or for inclusion within a custom database group. For example, because **elastic pools** are not yet directly supported using PowerShell APIs, you can create a custom database target and custom database collection target which encompasses all the databases in the pool.

Set the following variables to reflect the desired database information:

```
$databaseName = "{Database Name}"
$databaseServerName = "{Server Name}"
New-AzureSqlJobTarget -DatabaseName $databaseName -ServerName $databaseServerName
```

## To create a custom database collection target

Use the [New-AzureSqlJobTarget](#) cmdlet to define a custom database collection target to enable execution across multiple defined database targets. After creating a database group, databases can be associated with the custom collection target.

Set the following variables to reflect the desired custom collection target configuration:

```
$customCollectionName = "{Custom Database Collection Name}"
New-AzureSqlJobTarget -CustomCollectionName $customCollectionName
```

## To add databases to a custom database collection target

To add a database to a specific custom collection use the [Add-AzureSqlJobChildTarget](#) cmdlet.

```
$databaseServerName = "{Database Server Name}"
$databaseName = "{Database Name}"
$customCollectionName = "{Custom Database Collection Name}"
Add-AzureSqlJobChildTarget -CustomCollectionName $customCollectionName -DatabaseName $databaseName -ServerName
$databaseServerName
```

## Review the databases within a custom database collection target

Use the [Get-AzureSqlJobTarget](#) cmdlet to retrieve the child databases within a custom database collection

target.

```
$customCollectionName = "{Custom Database Collection Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
$childTargets = Get-AzureSqlJobTarget -ParentTargetId $target.TargetId
Write-Output $childTargets
```

### Create a job to execute a script across a custom database collection target

Use the [New-AzureSqlJob](#) cmdlet to create a job against a group of databases defined by a custom database collection target. Elastic Database jobs will expand the job into multiple child jobs each corresponding to a database associated with the custom database collection target and ensure that the script is executed against each database. Again, it is important that scripts are idempotent to be resilient to retries.

```
$jobName = "{Job Name}"
$scriptName = "{Script Name}"
$customCollectionName = "{Custom Collection Name}"
$credentialName = "{Credential Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
$job = New-AzureSqlJob -JobName $jobName -CredentialName $credentialName -ContentName $scriptName -TargetId
$target.TargetId
Write-Output $job
```

## Data collection across databases

You can use a job to execute a query across a group of databases and send the results to a specific table. The table can be queried after the fact to see the query's results from each database. This provides an asynchronous method to execute a query across many databases. Failed attempts are handled automatically via retries.

The specified destination table will be automatically created if it does not yet exist. The new table matches the schema of the returned result set. If a script returns multiple result sets, Elastic Database jobs will only send the first to the destination table.

The following PowerShell script executes a script and collects its results into a specified table. This script assumes that a T-SQL script has been created which outputs a single result set and that a custom database collection target has been created.

This script uses the [Get-AzureSqlJobTarget](#) cmdlet. Set the parameters for script, credentials, and execution target:

```
$jobName = "{Job Name}"
$scriptName = "{Script Name}"
$executionCredentialName = "{Execution Credential Name}"
$customCollectionName = "{Custom Collection Name}"
$destinationCredentialName = "{Destination Credential Name}"
$destinationServerName = "{Destination Server Name}"
$destinationDatabaseName = "{Destination Database Name}"
$destinationSchemaName = "{Destination Schema Name}"
$destinationTableName = "{Destination Table Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
```

### To create and start a job for data collection scenarios

This script uses the [Start-AzureSqlJobExecution](#) cmdlet.

```

$job = New-AzureSqlJob -JobName $jobName
-CredentialName $executionCredentialName
-ContentName $scriptName
-ResultSetDestinationServerName $destinationServerName
-ResultSetDestinationDatabaseName $destinationDatabaseName
-ResultSetDestinationSchemaName $destinationSchemaName
-ResultSetDestinationTableName $destinationTableName
-ResultSetDestinationCredentialName $destinationCredentialName
-TargetId $target.TargetId
Write-Output $job
$jobExecution = Start-AzureSqlJobExecution -JobName $jobName
Write-Output $jobExecution

```

## To schedule a job execution trigger

The following PowerShell script can be used to create a recurring schedule. This script uses a minute interval, but [New-AzureSqlJobSchedule](#) also supports -DayInterval, -HourInterval, -MonthInterval, and -WeekInterval parameters. Schedules that execute only once can be created by passing -OneTime.

Create a new schedule:

```

$scheduleName = "Every one minute"
$minuteInterval = 1
$startTime = (Get-Date).ToUniversalTime()
$schedule = New-AzureSqlJobSchedule
-MinuteInterval $minuteInterval
-ScheduleName $scheduleName
-StartTime $startTime
Write-Output $schedule

```

## To trigger a job executed on a time schedule

A job trigger can be defined to have a job executed according to a time schedule. The following PowerShell script can be used to create a job trigger.

Use [New-AzureSqlJobTrigger](#) and set the following variables to correspond to the desired job and schedule:

```

$jobName = "{Job Name}"
$scheduleName = "{Schedule Name}"
$jobTrigger = New-AzureSqlJobTrigger
-ScheduleName $scheduleName
-JobName $jobName
Write-Output $jobTrigger

```

## To remove a scheduled association to stop job from executing on schedule

To discontinue reoccurring job execution through a job trigger, the job trigger can be removed. Remove a job trigger to stop a job from being executed according to a schedule using the [Remove-AzureSqlJobTrigger cmdlet](#).

```

$jobName = "{Job Name}"
$scheduleName = "{Schedule Name}"
Remove-AzureSqlJobTrigger
-ScheduleName $scheduleName
-JobName $jobName

```

## Retrieve job triggers bound to a time schedule

The following PowerShell script can be used to obtain and display the job triggers registered to a particular time

schedule.

```
$scheduleName = "{Schedule Name}"
$jobTriggers = Get-AzureSqlJobTrigger -ScheduleName $scheduleName
Write-Output $jobTriggers
```

### To retrieve job triggers bound to a job

Use [Get-AzureSqlJobTrigger](#) to obtain and display schedules containing a registered job.

```
$jobName = "{Job Name}"
$jobTriggers = Get-AzureSqlJobTrigger -JobName $jobName
Write-Output $jobTriggers
```

## To create a data-tier application (DACPAC) for execution across databases

To create a DACPAC, see [Data-Tier applications](#). To deploy a DACPAC, use the [New-AzureSqlJobContent cmdlet](#). The DACPAC must be accessible to the service. It is recommended to upload a created DACPAC to Azure Storage and create a [Shared Access Signature](#) for the DACPAC.

```
$dcpacUri = "{Uri}"
$dcpacName = "{DACPAC Name}"
$dcpac = New-AzureSqlJobContent -DcpacUri $dcpacUri -ContentName $dcpacName
Write-Output $dcpac
```

### To update a data-tier application (DACPAC) for execution across databases

Existing DACPACs registered within Elastic Database Jobs can be updated to point to new URIs. Use the [Set-AzureSqlJobContentDefinition cmdlet](#) to update the DACPAC URI on an existing registered DACPAC:

```
$dcpacName = "{DACPAC Name}"
$newDcpacUri = "{Uri}"
$updatedDcpac = Set-AzureSqlJobDcpacDefinition -ContentName $dcpacName -DcpacUri $newDcpacUri
Write-Output $updatedDcpac
```

## To create a job to apply a data-tier application (DACPAC) across databases

After a DACPAC has been created within Elastic Database Jobs, a job can be created to apply the DACPAC across a group of databases. The following PowerShell script can be used to create a DACPAC job across a custom collection of databases:

```
$jobName = "{Job Name}"
$dcpacName = "{DACPAC Name}"
$customCollectionName = "{Custom Collection Name}"
$credentialName = "{Credential Name}"
$target = Get-AzureSqlJobTarget
-CustomCollectionName $customCollectionName
$job = New-AzureSqlJob
-JobName $jobName
-CredentialName $credentialName
-ContentName $dcpacName -TargetId $target.TargetId
Write-Output $job
```

## 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](#).

# Getting started with Elastic Database jobs

10/30/2018 • 13 minutes to read • [Edit Online](#)

## IMPORTANT

This article is for the customer-hosted version of *Elastic Database jobs*. Elastic Database jobs are being deprecated and replaced with new Azure-hosted **Elastic Database jobs**. For new jobs, use [the latest Elastic Database jobs](#). If you currently use the older customer-hosted jobs, see [Migrate to the new Elastic Database jobs](#) for directions and migration scripts to quickly upgrade to the latest version.

Elastic Database jobs (preview) for Azure SQL Database allows you to reliably execute T-SQL scripts that span multiple databases while automatically retrying and providing eventual completion guarantees. For more information about the Elastic Database job feature, see [Elastic jobs](#).

This article extends the sample found in [Getting started with Elastic Database tools](#). When completed, you learn how to create and manage jobs that manage a group of related databases. It is not required to use the Elastic Scale tools in order to take advantage of the benefits of Elastic jobs.

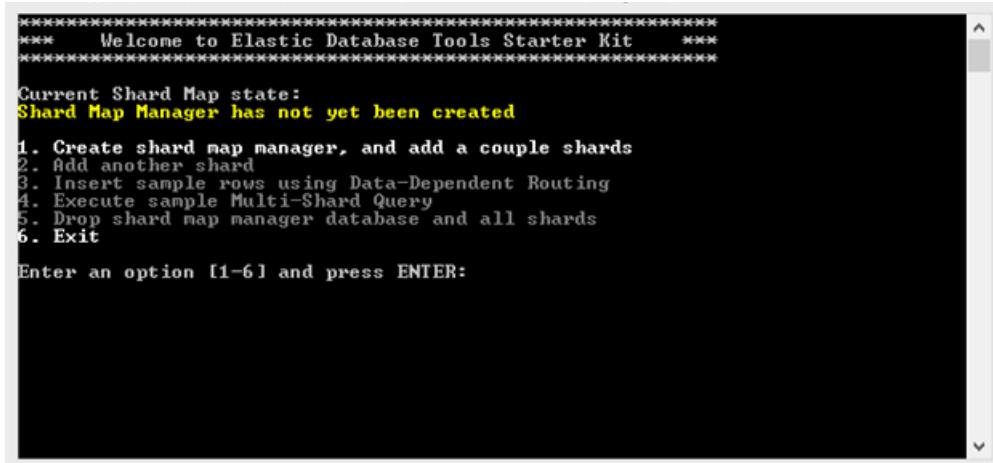
## Prerequisites

Download and run the [Getting started with Elastic Database tools sample](#).

## Create a shard map manager using the sample app

Here you create a shard map manager along with several shards, followed by insertion of data into the shards. If you already have shards set up 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. Follow the steps until step 7 in the section [Download and run the sample app](#). At the end of Step 7, you 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 this action four times. This inserts sample data rows in your shards.
3. The [Azure portal](#) should show three new databases:

Databases

SQL databases

**3 Databases** SQL V12

| DATABASE                                 | STATUS | PRICING TIER |
|--|--------|--------------|
| ElasticScaleStarterKit_Shard1            | Online | Basic        |
| ElasticScaleStarterKit_Shard0            | Online | Basic        |
| ElasticScaleStarterKit_ShardMapManagerDb | Online | Basic        |

At this point, we create a custom database collection that reflects all the databases in the shard map. This allows us to create and execute a job that adds a new table across shards.

Here we would usually create a shard map target, using the **New-AzureSqlJobTarget** cmdlet. The shard map manager database must be set as a database target and then the specific shard map is specified as a target. Instead, we are going to enumerate all the databases in the server and add the databases to the new custom collection with the exception of master database.

Creates a custom collection and add all databases in the server to the custom collection target with the exception of master.

```

$customCollectionName = "dbs_in_server"
New-AzureSqlJobTarget -CustomCollectionName $customCollectionName
$ResourceGroupName = "ddove_samples"
$ServerName = "samples"
$dbsinserver = Get-AzureRMSqlDatabase -ResourceGroupName $ResourceGroupName -ServerName $ServerName
$dbsinserver | %{
$currentdb = $_.DatabaseName
$ErrorActionPreference = "Stop"
Write-Output ""

Try
{
    New-AzureSqlJobTarget -ServerName $ServerName -DatabaseName $currentdb | Write-Output
}
Catch
{
    $ErrorMessage = $_.Exception.Message
    $ErrorCategory = $_.CategoryInfo.Reason

    if ($ErrorCategory -eq 'UniqueConstraintViolationException')
    {
        Write-Host $currentdb "is already a database target."
    }

    else
    {
        throw $_
    }
}

Try
{
    if ($currentdb -eq "master")
    {
        Write-Host $currentdb "will not be added custom collection target" $CustomCollectionName "."
    }

    else
    {
        Add-AzureSqlJobChildTarget -CustomCollectionName $CustomCollectionName -ServerName $ServerName -
DatabaseName $currentdb
        Write-Host $currentdb "was added to" $CustomCollectionName "."
    }
}

Catch
{
    $ErrorMessage = $_.Exception.Message
    $ErrorCategory = $_.CategoryInfo.Reason

    if ($ErrorCategory -eq 'UniqueConstraintViolationException')
    {
        Write-Host $currentdb "is already in the custom collection target" $CustomCollectionName"."
    }

    else
    {
        throw $_
    }
}
$ErrorActionPreference = "Continue"
}

```

## Create a T-SQL Script for execution across databases

```

$scriptName = "NewTable"
$scriptCommandText = "
IF NOT EXISTS (SELECT name FROM sys.tables WHERE name = 'Test')
BEGIN
    CREATE TABLE Test(
        TestId INT PRIMARY KEY IDENTITY,
        InsertionTime DATETIME2
    );
END
GO
INSERT INTO Test(InsertionTime) VALUES (sysutcdatetime());
GO"

$script = New-AzureSqlJobContent -ContentName $scriptName -CommandText $scriptCommandText
Write-Output $script

```

## Create the job to execute a script across the custom group of databases

```

$jobName = "create on server dbs"
$scriptName = "NewTable"
$customCollectionName = "dbs_in_server"
$credentialName = "ddove66"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
$job = New-AzureSqlJob -JobName $jobName -CredentialName $credentialName -ContentName $scriptName -TargetId
$target.TargetId
Write-Output $job

```

## Execute the job

The following PowerShell script can be used to execute an existing job:

Update the following variable to reflect the desired job name to have executed:

```

$jobName = "create on server dbs"
$jobExecution = Start-AzureSqlJobExecution -JobName $jobName
Write-Output $jobExecution

```

## Retrieve the state of a single job execution

Use the same **Get-AzureSqlJobExecution** cmdlet with the **IncludeChildren** parameter to view the state of child job executions, namely the specific state for each job execution against each database targeted by the job.

```

$jobExecutionId = "{Job Execution Id}"
$jobExecutions = Get-AzureSqlJobExecution -JobExecutionId $jobExecutionId -IncludeChildren
Write-Output $jobExecutions

```

## View the state across multiple job executions

The **Get-AzureSqlJobExecution** cmdlet has multiple optional parameters that can be used to display multiple job executions, filtered through the provided parameters. The following demonstrates some of the possible ways to use Get-AzureSqlJobExecution:

Retrieve all active top-level job executions:

```
Get-AzureSqlJobExecution
```

Retrieve all top-level job executions, including inactive job executions:

```
Get-AzureSqlJobExecution -IncludeInactive
```

Retrieve all child job executions of a provided job execution ID, including inactive job executions:

```
$parentJobExecutionId = "{Job Execution Id}"
Get-AzureSqlJobExecution -AzureSqlJobExecution -JobExecutionId $parentJobExecutionId -IncludeInactive -IncludeChildren
```

Retrieve all job executions created using a schedule / job combination, including inactive jobs:

```
$jobName = "{Job Name}"
$scheduleName = "{Schedule Name}"
Get-AzureSqlJobExecution -JobName $jobName -ScheduleName $scheduleName -IncludeInactive
```

Retrieve all jobs targeting a specified shard map, including inactive jobs:

```
$shardMapServerName = "{Shard Map Server Name}"
$shardMapDatabaseName = "{Shard Map Database Name}"
$shardMapName = "{Shard Map Name}"
$target = Get-AzureSqlJobTarget -ShardMapManagerDatabaseName $shardMapDatabaseName -ShardMapManagerServerName $shardMapServerName -ShardMapName $shardMapName
Get-AzureSqlJobExecution -TargetId $target.TargetId -IncludeInactive
```

Retrieve all jobs targeting a specified custom collection, including inactive jobs:

```
$customCollectionName = "{Custom Collection Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
Get-AzureSqlJobExecution -TargetId $target.TargetId -IncludeInactive
```

Retrieve the list of job task executions within a specific job execution:

```
$jobExecutionId = "{Job Execution Id}"
$jobTaskExecutions = Get-AzureSqlJobTaskExecution -JobExecutionId $jobExecutionId
Write-Output $jobTaskExecutions
```

Retrieve job task execution details:

The following PowerShell script can be used to view the details of a job task execution, which is particularly useful when debugging execution failures.

```
$jobTaskExecutionId = "{Job Task Execution Id}"
$jobTaskExecution = Get-AzureSqlJobTaskExecution -JobTaskExecutionId $jobTaskExecutionId
Write-Output $jobTaskExecution
```

## Retrieve failures within job task executions

The JobTaskExecution object includes a property for the Lifecycle of the task along with a Message property. If a job task execution failed, the Lifecycle property is set to *Failed* and the Message property is set to the resulting

exception message and its stack. If a job did not succeed, it is important to view the details of job tasks that did not succeed for a given job.

```
$jobExecutionId = "{Job Execution Id}"
$jobTaskExecutions = Get-AzureSqlJobTaskExecution -JobExecutionId $jobExecutionId
Foreach($jobTaskExecution in $jobTaskExecutions)
{
    if($jobTaskExecution.Lifecycle -ne 'Succeeded')
    {
        Write-Output $jobTaskExecution
    }
}
```

## Waiting for a job execution to complete

The following PowerShell script can be used to wait for a job task to complete:

```
$jobExecutionId = "{Job Execution Id}"
Wait-AzureSqlJobExecution -JobExecutionId $jobExecutionId
```

## Create a custom execution policy

Elastic Database jobs supports creating custom execution policies that can be applied when starting jobs.

Execution policies currently allow for defining:

- Name: Identifier for the execution policy.
- Job Timeout: Total time before a job is canceled by Elastic Database Jobs.
- Initial Retry Interval: Interval to wait before first retry.
- Maximum Retry Interval: Cap of retry intervals to use.
- Retry Interval Backoff Coefficient: Coefficient used to calculate the next interval between retries. The following formula is used: (Initial Retry Interval) \* Math.pow((Interval Backoff Coefficient), (Number of Retries) - 2).
- Maximum Attempts: The maximum number of retry attempts to perform within a job.

The default execution policy uses the following values:

- Name: Default execution policy
- Job Timeout: 1 week
- Initial Retry Interval: 100 milliseconds
- Maximum Retry Interval: 30 minutes
- Retry Interval Coefficient: 2
- Maximum Attempts: 2,147,483,647

Create the desired execution policy:

```
$executionPolicyName = "{Execution Policy Name}"
$initialRetryInterval = New-TimeSpan -Seconds 10
$jobTimeout = New-TimeSpan -Minutes 30
$maximumAttempts = 999999
$maximumRetryInterval = New-TimeSpan -Minutes 1
$retryIntervalBackoffCoefficient = 1.5
$executionPolicy = New-AzureSqlJobExecutionPolicy -ExecutionPolicyName $executionPolicyName -
InitialRetryInterval $initialRetryInterval -JobTimeout $jobTimeout -MaximumAttempts $maximumAttempts -
MaximumRetryInterval $maximumRetryInterval -RetryIntervalBackoffCoefficient $retryIntervalBackoffCoefficient
Write-Output $executionPolicy
```

## Update a custom execution policy

Update the desired execution policy to update:

```
$executionPolicyName = "{Execution Policy Name}"
$initialRetryInterval = New-TimeSpan -Seconds 15
$jobTimeout = New-TimeSpan -Minutes 30
$maximumAttempts = 999999
$maximumRetryInterval = New-TimeSpan -Minutes 1
$retryIntervalBackoffCoefficient = 1.5
$updatedExecutionPolicy = Set-AzureSqlJobExecutionPolicy -ExecutionPolicyName $executionPolicyName -
InitialRetryInterval $initialRetryInterval -JobTimeout $jobTimeout -MaximumAttempts $maximumAttempts -
MaximumRetryInterval $maximumRetryInterval -RetryIntervalBackoffCoefficient $retryIntervalBackoffCoefficient
Write-Output $updatedExecutionPolicy
```

## Cancel a job

Elastic Database Jobs supports jobs cancellation requests. If Elastic Database Jobs detects a cancellation request for a job currently being executed, it attempts to stop the job.

There are two different ways that Elastic Database Jobs can perform a cancellation:

1. Canceling Currently Executing Tasks: If a cancellation is detected while a task is currently running, a cancellation is attempted within the currently executing aspect of the task. For example: If there is a long running query currently being performed when a cancellation is attempted, there is an attempt to cancel the query.
2. Canceling Task Retries: If a cancellation is detected by the control thread before a task is launched for execution, the control thread avoids launching the task and declare the request as canceled.

If a job cancellation is requested for a parent job, the cancellation request is honored for the parent job and for all of its child jobs.

To submit a cancellation request, use the **Stop-AzureSqlJobExecution** cmdlet and set the **JobExecutionId** parameter.

```
$jobExecutionId = "{Job Execution Id}"
Stop-AzureSqlJobExecution -JobExecutionId $jobExecutionId
```

## Delete a job by name and the job's history

Elastic Database jobs supports asynchronous deletion of jobs. A job can be marked for deletion and the system deletes the job and all its job history after all job executions have completed for the job. The system does not automatically cancel active job executions.

Instead, Stop-AzureSqlJobExecution must be invoked to cancel active job executions.

To trigger job deletion, use the **Remove-AzureSqlJob** cmdlet and set the **JobName** parameter.

```
$jobName = "{Job Name}"
Remove-AzureSqlJob -JobName $jobName
```

## Create a custom database target

Custom database targets can be defined in Elastic Database jobs which can be used either for execution directly or for inclusion within a custom database group. Since **elastic pools** are not yet directly supported via the PowerShell APIs, you simply create a custom database target and custom database collection target which

encompasses all the databases in the pool.

Set the following variables to reflect the desired database information:

```
$databaseName = "{Database Name}"
$databaseServerName = "{Server Name}"
New-AzureSqlJobDatabaseTarget -DatabaseName $databaseName -ServerName $databaseServerName
```

## Create a custom database collection target

A custom database collection target can be defined to enable execution across multiple defined database targets. After a database group is created, databases can be associated to the custom collection target.

Set the following variables to reflect the desired custom collection target configuration:

```
$customCollectionName = "{Custom Database Collection Name}"
New-AzureSqlJobTarget -CustomCollectionName $customCollectionName
```

### Add databases to a custom database collection target

Database targets can be associated with custom database collection targets to create a group of databases. Whenever a job is created that targets a custom database collection target, it is expanded to target the databases associated to the group at the time of execution.

Add the desired database to a specific custom collection:

```
$serverName = "{Database Server Name}"
$databaseName = "{Database Name}"
$customCollectionName = "{Custom Database Collection Name}"
Add-AzureSqlJobChildTarget -CustomCollectionName $customCollectionName -DatabaseName $databaseName -
ServerName $databaseServerName
```

### Review the databases within a custom database collection target

Use the **Get-AzureSqlJobTarget** cmdlet to retrieve the child databases within a custom database collection target.

```
$customCollectionName = "{Custom Database Collection Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
$childTargets = Get-AzureSqlJobTarget -ParentTargetId $target.TargetId
Write-Output $childTargets
```

### Create a job to execute a script across a custom database collection target

Use the **New-AzureSqlJob** cmdlet to create a job against a group of databases defined by a custom database collection target. Elastic Database jobs expands the job into multiple child jobs each corresponding to a database associated with the custom database collection target and ensure that the script is executed against each database. Again, it is important that scripts are idempotent to be resilient to retries.

```
$jobName = "{Job Name}"
$scriptName = "{Script Name}"
$customCollectionName = "{Custom Collection Name}"
$credentialName = "{Credential Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
$job = New-AzureSqlJob -JobName $jobName -CredentialName $credentialName -ContentName $scriptName -TargetId
$target.TargetId
Write-Output $job
```

## Data collection across databases

**Elastic Database jobs** supports executing a query across a group of databases and sends the results to a specified database's table. The table can be queried after the fact to see the query's results from each database. This provides an asynchronous mechanism to execute a query across many databases. Failure cases such as one of the databases being temporarily unavailable are handled automatically via retries.

The specified destination table is automatically created if it does not yet exist, matching the schema of the returned result set. If a script execution returns multiple result sets, Elastic Database jobs only sends the first one to the provided destination table.

The following PowerShell script can be used to execute a script collecting its results into a specified table. This script assumes that a T-SQL script has been created which outputs a single result set and a custom database collection target has been created.

Set the following to reflect the desired script, credentials, and execution target:

```
$jobName = "{Job Name}"
$scriptName = "{Script Name}"
$executionCredentialName = "{Execution Credential Name}"
$customCollectionName = "{Custom Collection Name}"
$destinationCredentialName = "{Destination Credential Name}"
$destinationServerName = "{Destination Server Name}"
$destinationDatabaseName = "{Destination Database Name}"
$destinationSchemaName = "{Destination Schema Name}"
$destinationTableName = "{Destination Table Name}"
$target = Get-AzureSqlJobTarget -CustomCollectionName $customCollectionName
```

### Create and start a job for data collection scenarios

```
$job = New-AzureSqlJob -JobName $jobName -CredentialName $executionCredentialName -ContentName $scriptName -
ResultSetDestinationServerName $destinationServerName -ResultSetDestinationDatabaseName
$destinationDatabaseName -ResultSetDestinationSchemaName $destinationSchemaName -
ResultSetDestinationTableName $destinationTableName -ResultSetDestinationCredentialName
$destinationCredentialName -TargetId $target.TargetId
Write-Output $job
$jobExecution = Start-AzureSqlJobExecution -JobName $jobName
Write-Output $jobExecution
```

## Create a schedule for job execution using a job trigger

The following PowerShell script can be used to create a reoccurring schedule. This script uses a one minute interval, but New-AzureSqlJobSchedule also supports -DayInterval, -HourInterval, -MonthInterval, and -WeekInterval parameters. Schedules that execute only once can be created by passing -OneTime.

Create a new schedule:

```
$scheduleName = "Every one minute"
$minuteInterval = 1
$startTime = (Get-Date).ToUniversalTime()
$schedule = New-AzureSqlJobSchedule -MinuteInterval $minuteInterval -ScheduleName $scheduleName -StartTime
$startTime
Write-Output $schedule
```

### Create a job trigger to have a job executed on a time schedule

A job trigger can be defined to have a job executed according to a time schedule. The following PowerShell script can be used to create a job trigger.

Set the following variables to correspond to the desired job and schedule:

```
$jobName = "{Job Name}"
$scheduleName = "{Schedule Name}"
$jobTrigger = New-AzureSqlJobTrigger -ScheduleName $scheduleName -JobName $jobName
Write-Output $jobTrigger
```

### Remove a scheduled association to stop job from executing on schedule

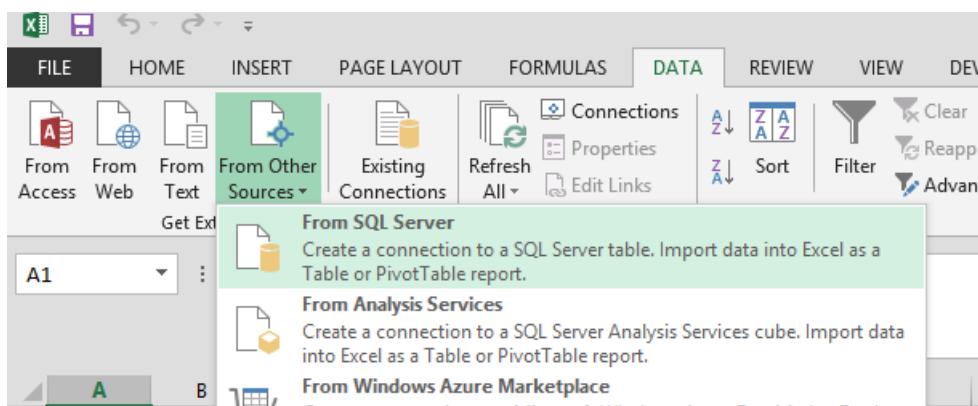
To discontinue reoccurring job execution through a job trigger, the job trigger can be removed. Remove a job trigger to stop a job from being executed according to a schedule using the **Remove-AzureSqlJobTrigger** cmdlet.

```
$jobName = "{Job Name}"
$scheduleName = "{Schedule Name}"
Remove-AzureSqlJobTrigger -ScheduleName $scheduleName -JobName $jobName
```

## 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.

## Next steps

You can now use Excel's data functions. 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. 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. However, at this time this feature is

available only on Premium and Business Critical databases and elastic pools as an end point, but the shards can be of any service tier.

For pricing information see [SQL Database Pricing Details](#).

## 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](#).

# Uninstall Elastic Database jobs components

10/30/2018 • 2 minutes to read • [Edit Online](#)

## IMPORTANT

This article is for the customer-hosted version of *Elastic Database jobs*. Elastic Database jobs are being deprecated and replaced with new Azure-hosted **Elastic Database jobs**. For new jobs, use the latest **Elastic Database jobs**. If you currently use the older customer-hosted jobs, see [Migrate to the new Elastic Database jobs](#) for directions and migration scripts to quickly upgrade to the latest version.

**Elastic Database jobs** components can be uninstalled using either the Azure portal or PowerShell.

## Uninstall Elastic Database jobs components using the Azure portal

1. Open the [Azure portal](#).
2. Navigate to the subscription that contains **Elastic Database jobs** components, namely the subscription in which Elastic Database jobs components were installed.
3. Click **Browse** and click **Resource groups**.
4. Select the resource group named "`__ElasticDatabaseJob`".
5. Delete the resource group.

## Uninstall Elastic Database jobs components using PowerShell

1. Launch a Microsoft Azure PowerShell command window and navigate to the tools sub-directory under the `Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x` folder: Type **cd tools**.

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x*> cd tools
```

2. Execute the `\UninstallElasticDatabaseJobs.ps1` PowerShell script.

```
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x\tools> Unblock-File .\UninstallElasticDatabaseJobs.ps1  
PS C:\*Microsoft.Azure.SqlDatabase.Jobs.x.x.xxxx.x\tools>.\UninstallElasticDatabaseJobs.ps1
```

Or simply, execute the following script, assuming default values where used on installation of the components:

```
$ResourceGroupName = "__ElasticDatabaseJob"  
Switch-AzureMode AzureResourceManager  
  
$resourceGroup = Get-AzureResourceGroup -Name $ResourceGroupName  
if(!$resourceGroup)  
{  
    Write-Host "The Azure Resource Group: $ResourceGroupName has already been deleted. Elastic database  
job components are uninstalled."  
    return  
}  
  
Write-Host "Removing the Azure Resource Group: $ResourceGroupName. This may take a few minutes."  
Remove-AzureResourceGroup -Name $ResourceGroupName -Force  
Write-Host "Completed removing the Azure Resource Group: $ResourceGroupName. Elastic database job  
components are now uninstalled."
```

## Next steps

To re-install Elastic Database jobs, see [Installing the Elastic Database job service](#)

For an overview of Elastic Database jobs, see [Elastic Database jobs overview](#).

# Azure SQL Database elastic query overview (preview)

10/30/2018 • 9 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`.

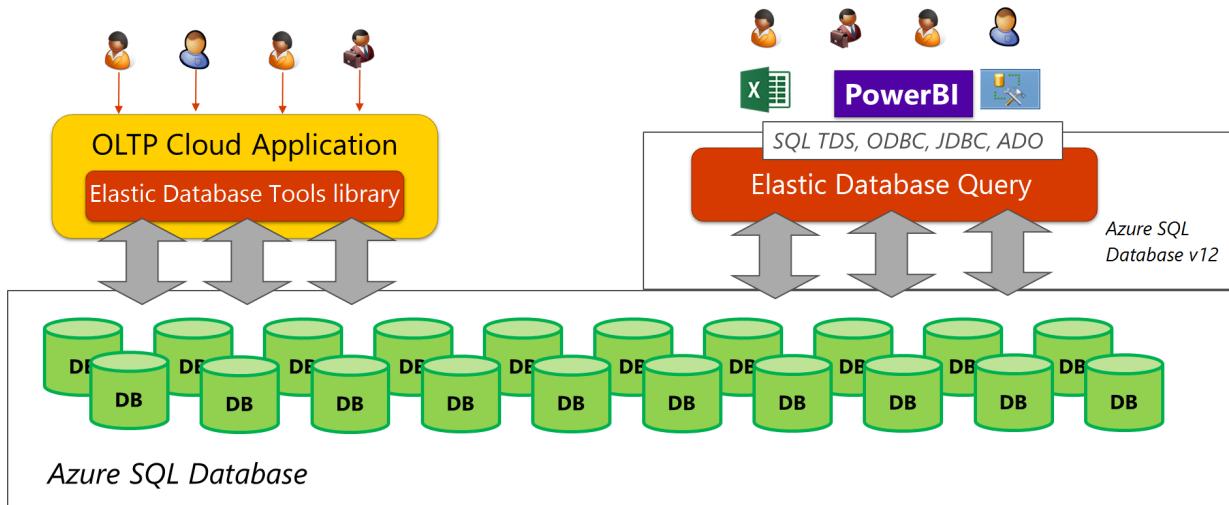
### Flexibility

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.

#### 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 SQL Data Warehouse](#).

## 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.

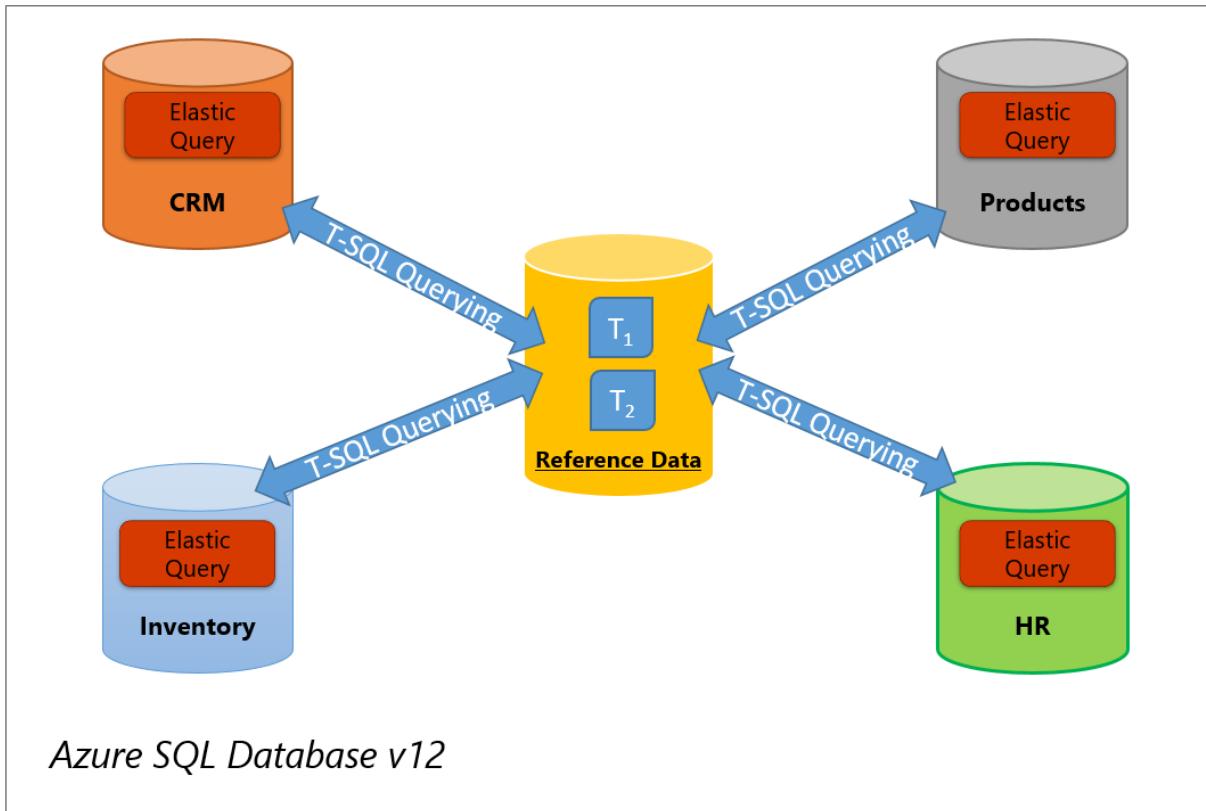
#### IMPORTANT

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

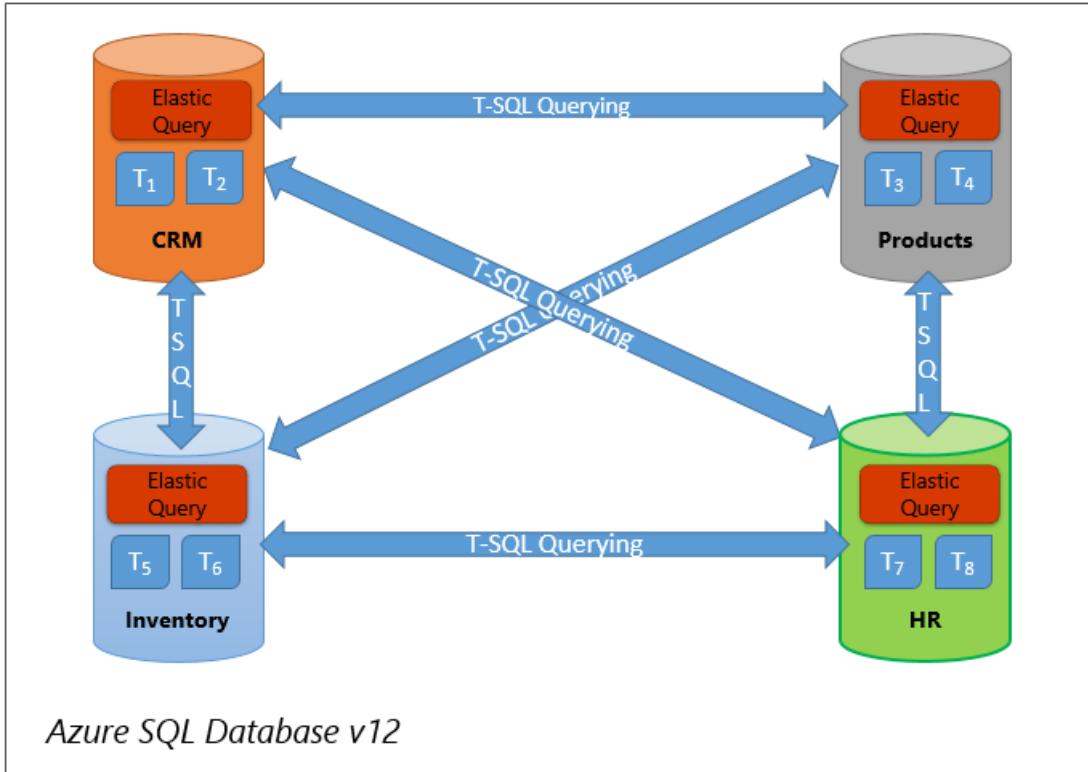
and T2 remotely from other databases, as shown in the figure. Use topology 1 if reference tables are small or remote queries into reference table have selective predicates.

**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:

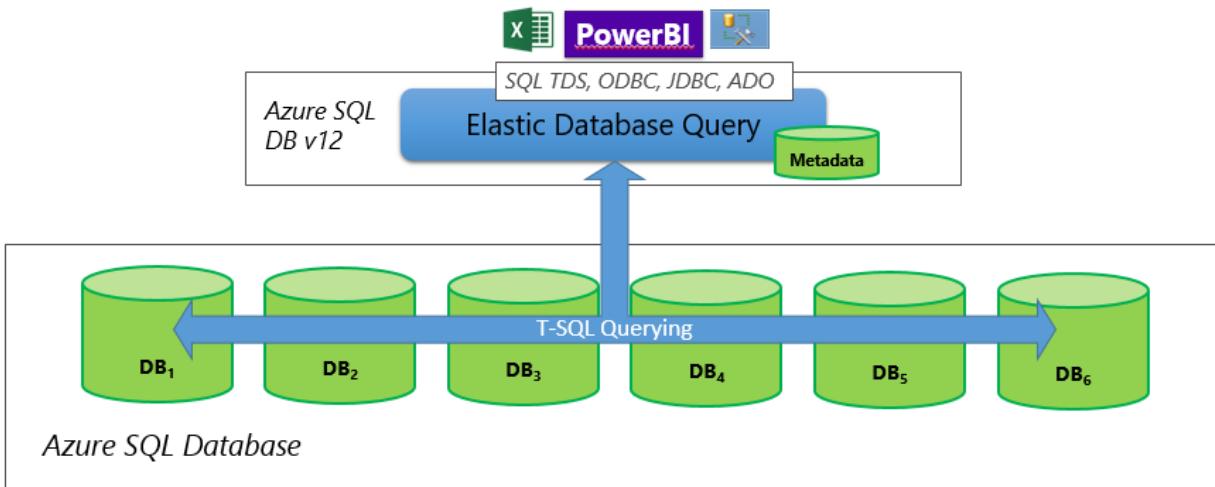
- `CREATE MASTER KEY` mymasterkey
- `CREATE DATABASE SCOPED CREDENTIAL` mycredential
- `CREATE/DROP EXTERNAL DATA SOURCE` mydatasource of type **RDBMS**
- `CREATE/DROP EXTERNAL TABLE` mytable

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 \(sharding\)](#).

## 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 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.

## Feedback

Share feedback on your experience with elastic queries with us below, on the MSDN forums, or on Stackoverflow. 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.

# Get started with cross-database queries (vertical partitioning) (preview)

10/30/2018 • 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 logical 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](#).

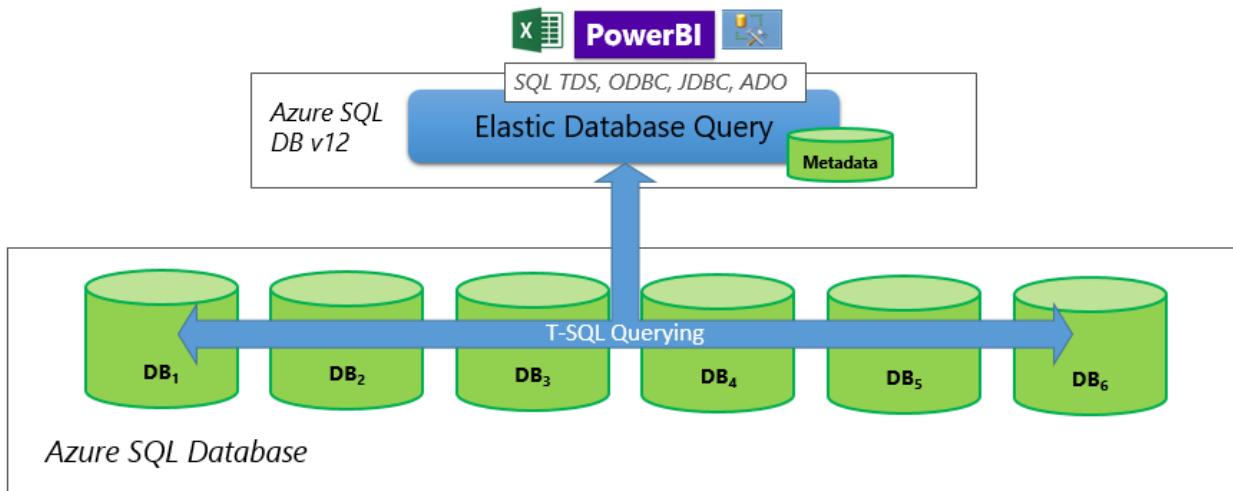
## Next steps

- 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)

10/30/2018 • 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

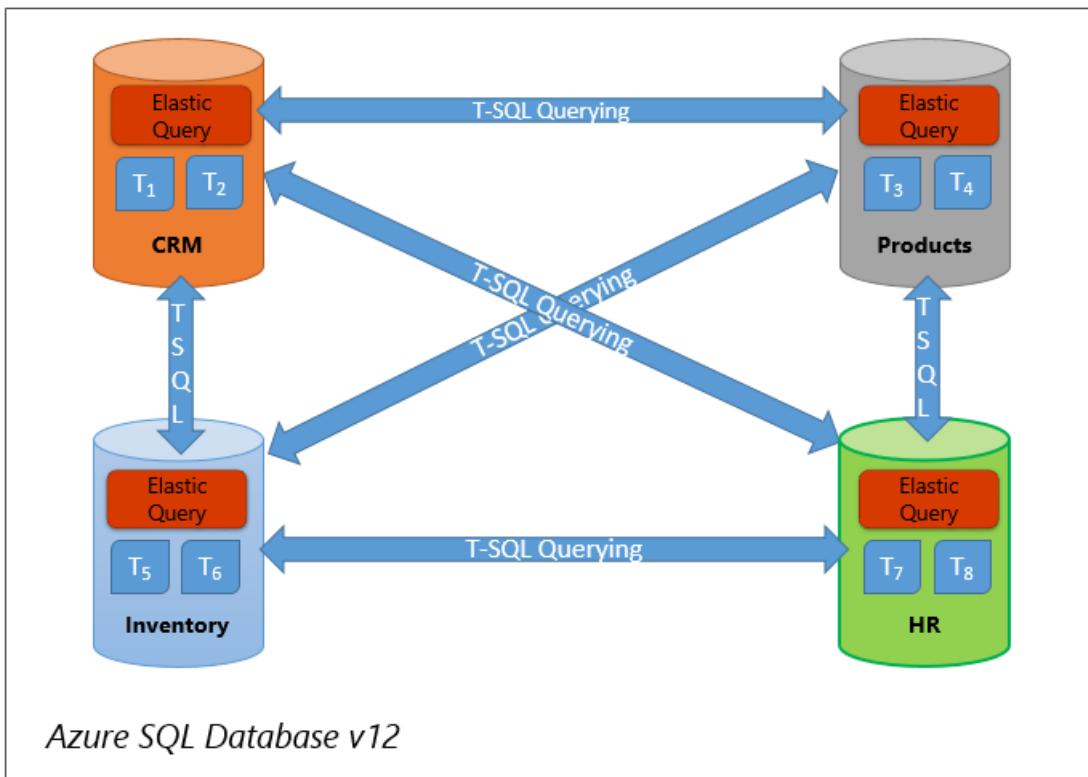
- Ensure that the elastic query endpoint database has been given access to the shardmap database and all shards through the SQL DB firewalls.
- Validate or enforce the data distribution defined by the external table. If your actual data distribution is different from the distribution specified in your 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)

10/30/2018 • 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 logical 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

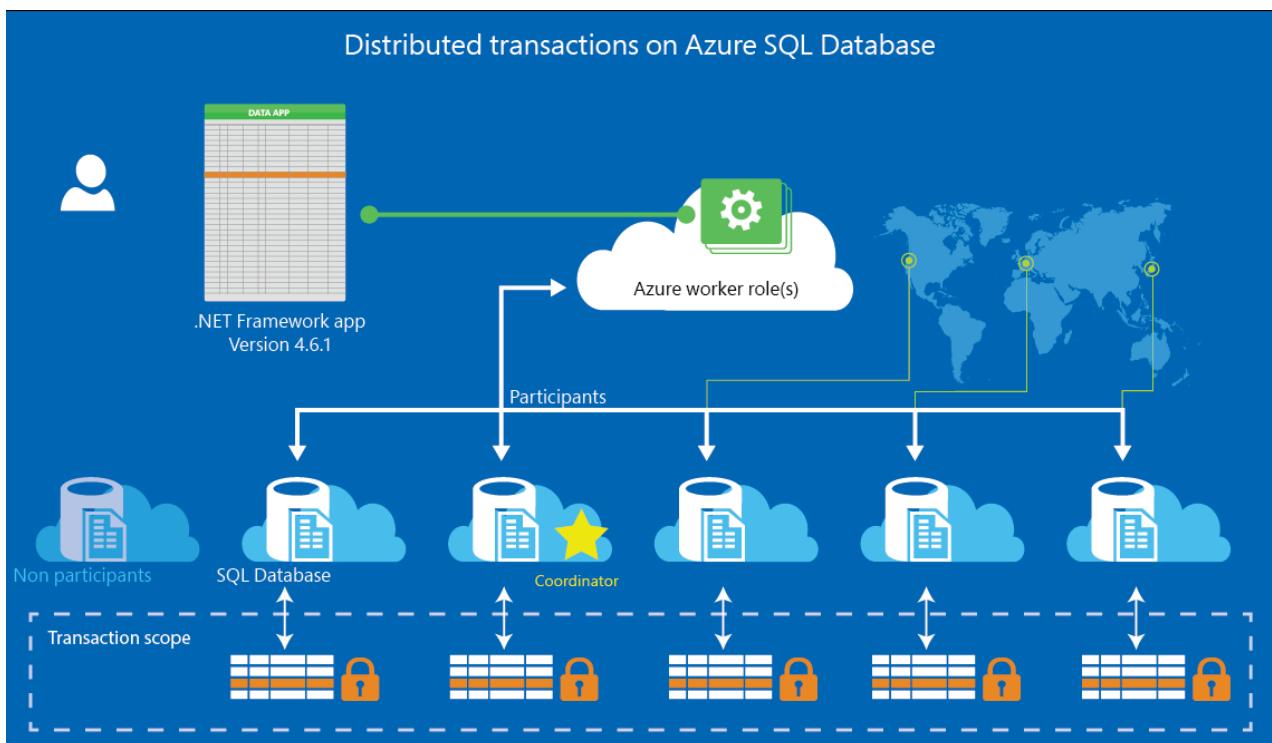
- For an overview of elastic query, see [Elastic query overview](#).
- For a vertical partitioning tutorial, see [Getting started with cross-database query \(vertical partitioning\)](#).
- For a horizontal partitioning (sharding) tutorial, see [Getting started with elastic query for horizontal partitioning \(sharding\)](#).
- 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.

# Distributed transactions across cloud databases

9/25/2018 • 7 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

Elastic database transactions are supported across different logical servers in Azure SQL Database. When transactions cross logical 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 logical servers, a communication relationship needs to be in place for any pair of logical servers.

Use the following PowerShell cmdlets to manage cross-server communication relationships for elastic database transactions:

- **New-AzureRmSqlServerCommunicationLink**: Use this cmdlet to create a new communication relationship between two logical servers in Azure SQL DB. The relationship is symmetric which means both servers can initiate transactions with the other server.
- **Get-AzureRmSqlServerCommunicationLink**: Use this cmdlet to retrieve existing communication relationships and their properties.
- **Remove-AzureRmSqlServerCommunicationLink**: 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-tenant SaaS database tenancy patterns

9/26/2018 • 12 minutes to read • [Edit Online](#)

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.

This article describes alternative tenancy models.

## 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.

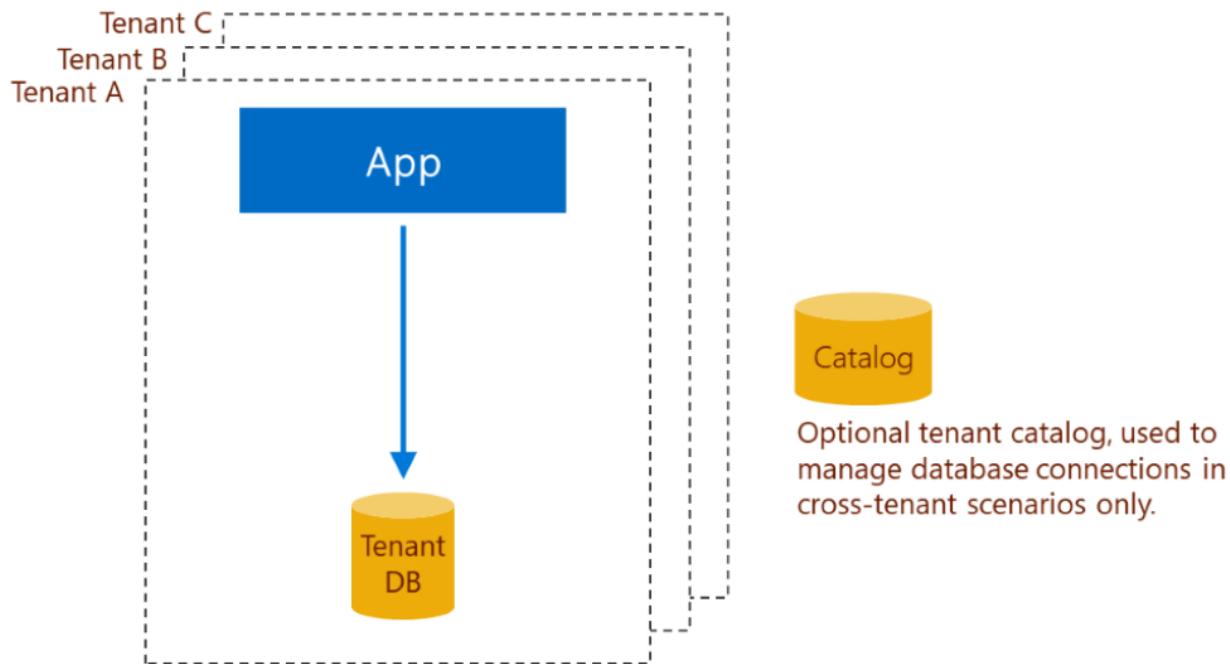
- **Customizability:** Ease of supporting schema customizations that are either tenant-specific or tenant class-specific.

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.

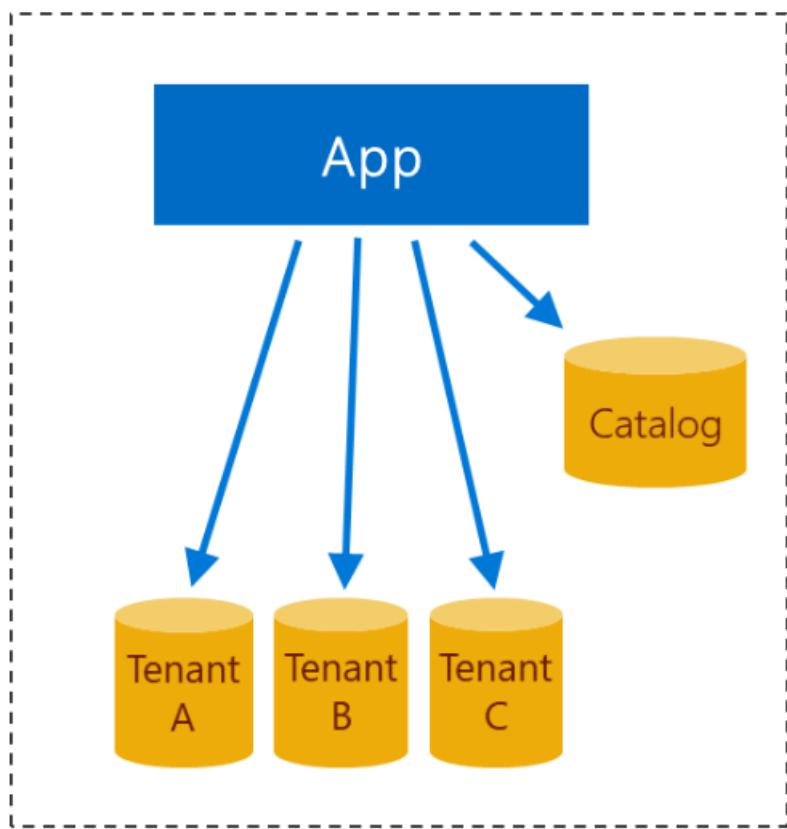
### Vendor management

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 URIs. 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.



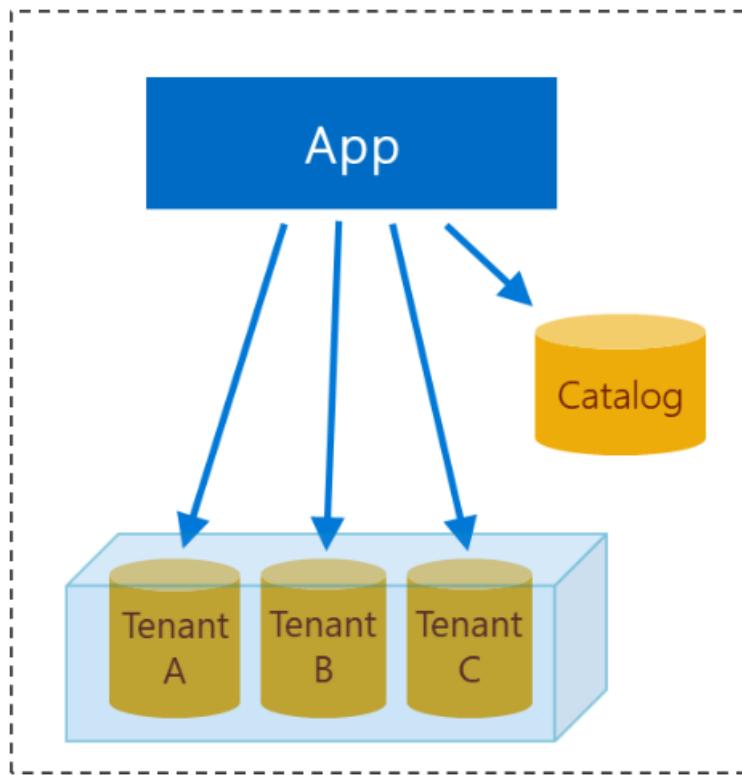
#### **Customize for a tenant**

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 database 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 Log Analytics. 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.

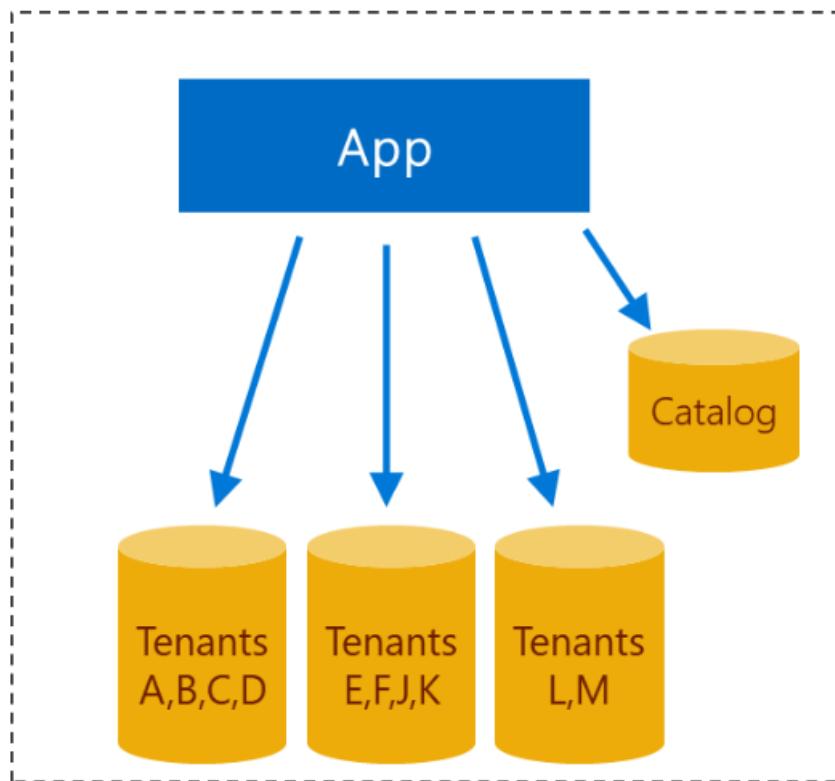
## F. Multi-tenant app with a single multi-tenant database

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<br>1-100s                                 | Very high<br>1-100,000s                           | Unlimited<br>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

9/24/2018 • 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](#) [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

Learning from MYOB: Design patterns for SaaS applications on Azure SQL Database - BRK3120

# Learning from MYOB: Design patterns for SaaS applications on SQL Database

Bill.Gibson@microsoft.com  
Principal Program Manager,  
Azure SQL Database

MORE VIDEOS Microsoft

0:00 / 12:156

[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.

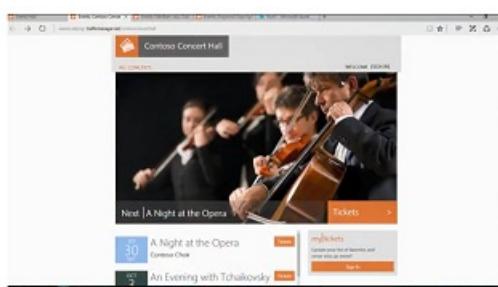
### 3. Agenda, 0:04:09



#### 4. Multi-tenant web app, 0:05:00



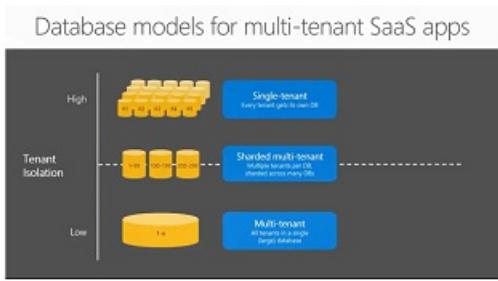
#### 5. App web form in action, 0:05:39



#### 6. Per-tenant cost (scale, isolation, recovery), 0:06:58

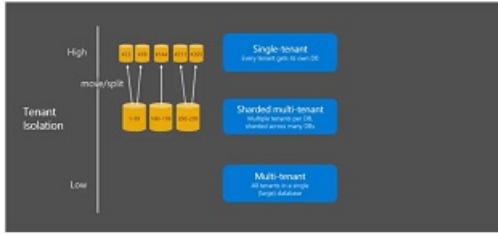


#### 7. Database models for multi-tenant: pros and cons, 0:09:52



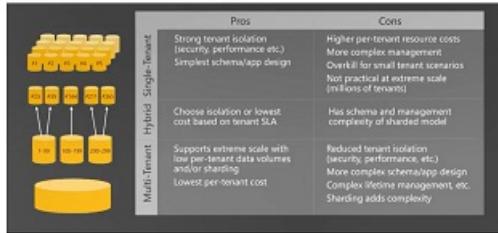
#### 8. Hybrid model blends benefits of MT/ST, 0:12:29

## Hybrid model blends benefits of MT/ST

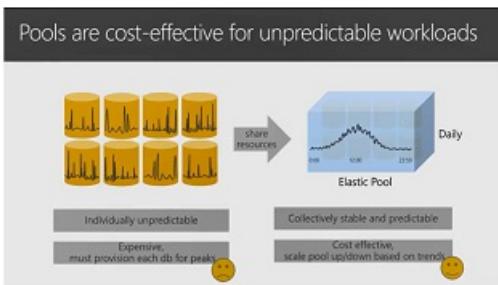


## 9. Single-tenant vs multi-tenant: pros and cons, 0:13:11

Single-tenant vs multi-tenant: pros and cons



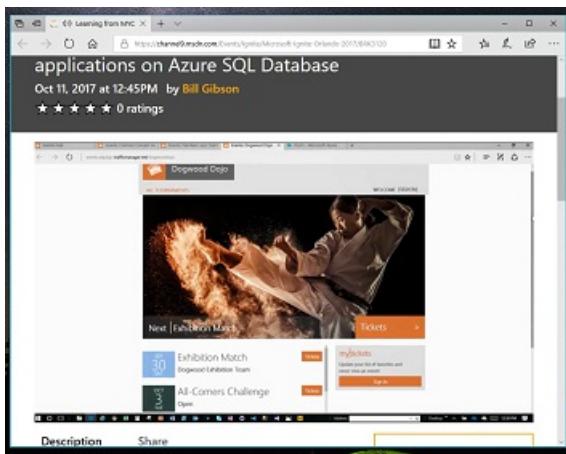
## 10. Pools are cost-effective for unpredictable workloads, 0:17:49



## 11. Demo of database-per-tenant and hybrid ST/MT, 0:19:59



## 12. Live app form showing Dojo, 0:20:10



### 13. MYOB and not a DBA in sight, 0:25:06

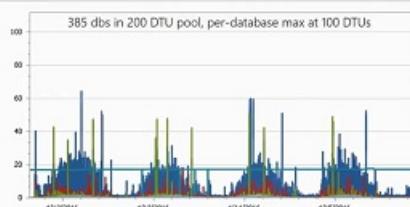
#### MYOB

- 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

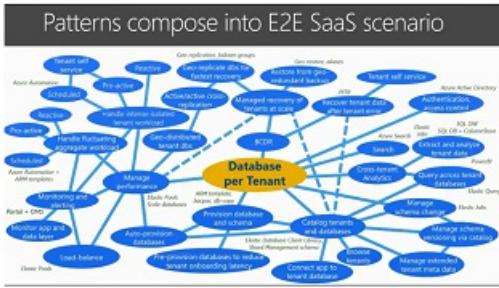


### 15. Learning from MYOB and other ISVs, 0:31:25

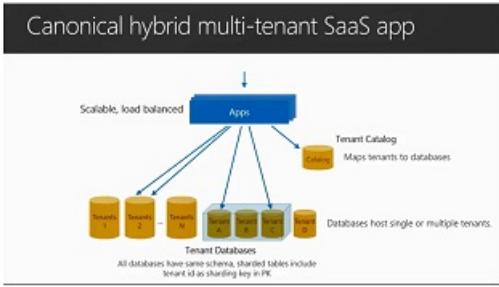
Learning from MYOB and other ISVs

SaaS patterns that address key design  
and management scenarios that are  
important at scale

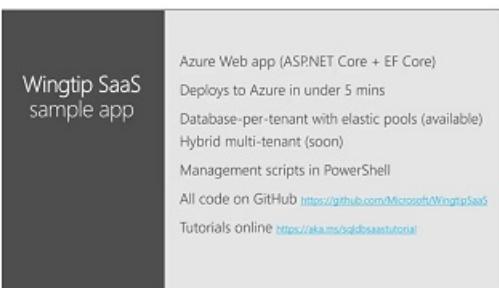
### 16. Patterns compose into E2E SaaS scenario, 0:31:42



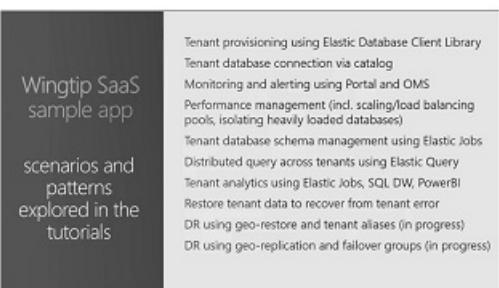
## 17. Canonical hybrid multi-tenant SaaS app, 0:46:04



## 18. Wingtip SaaS sample app, 0:48:01



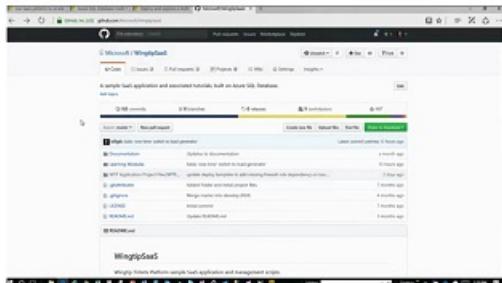
## 19. Scenarios and patterns explored in the tutorials, 0:49:00



## 20. Demo of tutorials and GitHub repository, 0:50:12



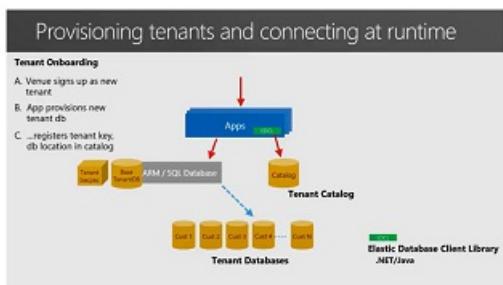
## 21. Github repo Microsoft/WingtipSaaS, 0:50:32



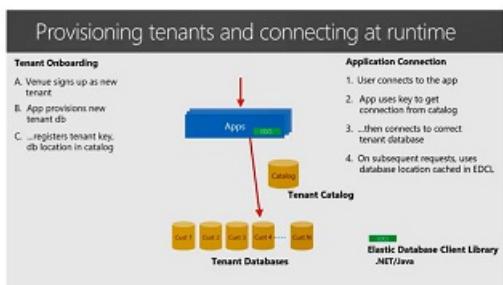
## 22. Exploring the patterns, 0:56:15



## 23. Provisioning tenants and onboarding, 0:56:19



## 24. Provisioning tenants and application connection, 0:57:52

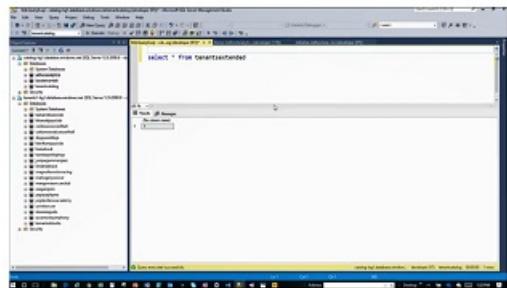


## 25. Demo of management scripts provisioning a single tenant, 0:59:36

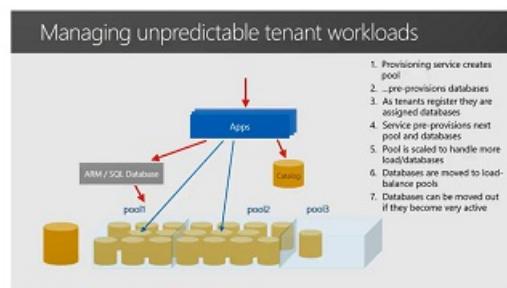


26. PowerShell to provision and catalog, 0:59:56

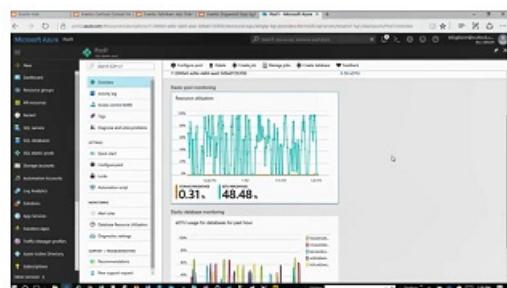
27. T-SQL SELECT \* FROM TenantsExtended, 1:03:25



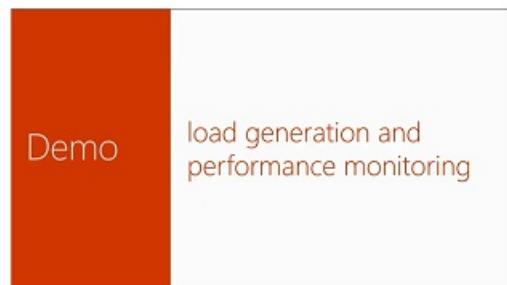
28. Managing unpredictable tenant workloads, 1:03:34



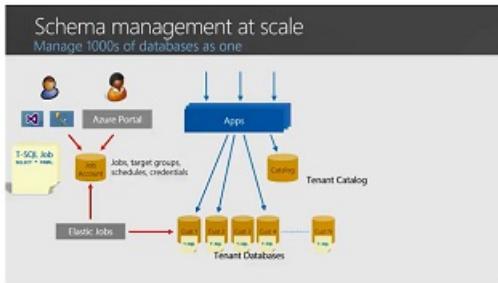
29. Elastic pool monitoring, 1:06:32



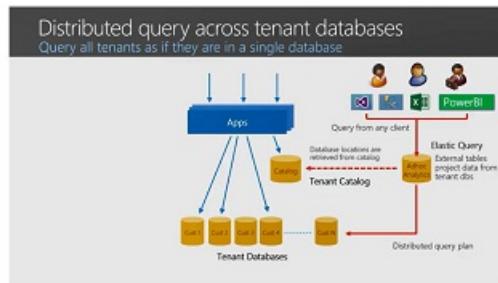
30. Load generation and performance monitoring, 1:09:37



### 31. Schema management at scale, 1:09:40



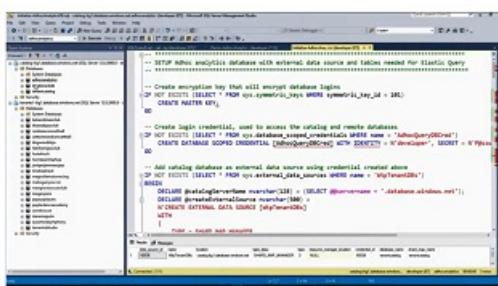
### 32. Distributed query across tenant databases, 1:11:18



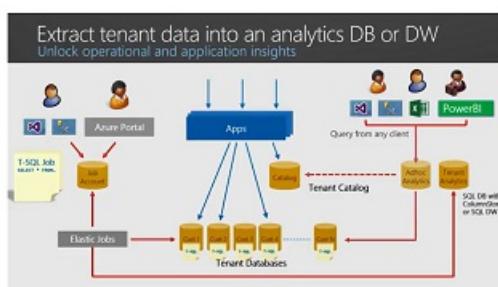
### 33. Demo of ticket generation, 1:12:28



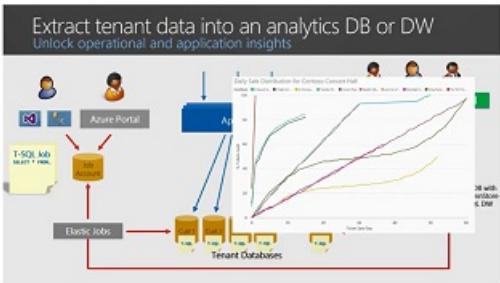
### 34. SSMS adhoc analytics, 1:12:35



### 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/sqldbsaastutorial>
- 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

10/30/2018 • 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**.

**Framework.** For more information, see [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).

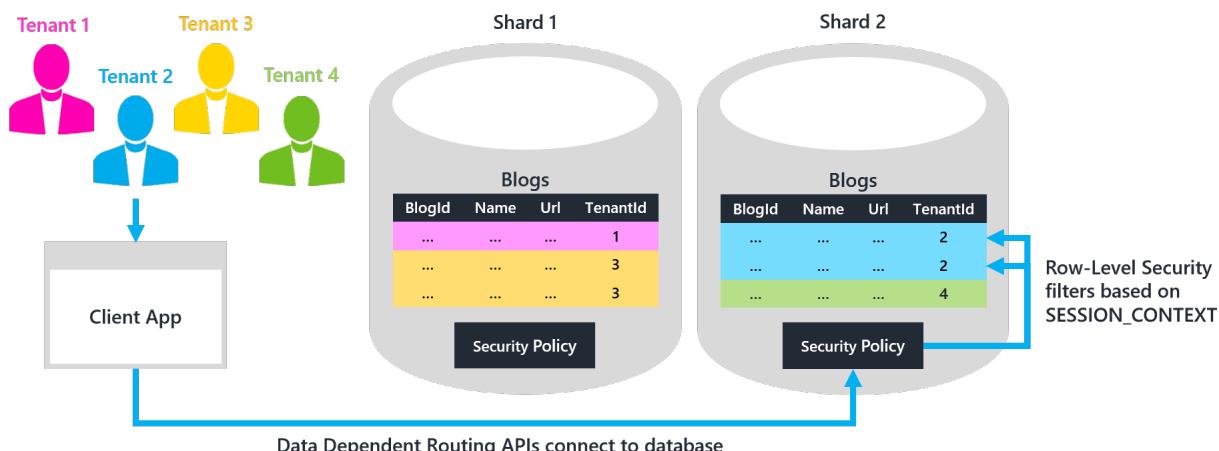
- **Elastic database tools** enable developers to scale out the data tier with standard sharding practices, by using .NET libraries and Azure service templates. Managing shards by using the [Elastic Database Client Library](#) helps automate and streamline many of the infrastructural tasks typically 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 SqlClient, 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](#).

# SQL Database application development overview

9/24/2018 • 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.

## TIP

For a tutorial showing you how to create a server, create a server-based firewall, view server properties, connect using SQL Server Management Studio, query the master database, create a sample database and a blank database, query database properties, connect using SQL Server Management Studio, and query the sample database, see [Get Started Tutorial](#).

## Language and platform

There are code samples available for various programming languages and platforms. You can find links to the code samples at:

- More information: [Connection libraries for SQL Database and SQL Server](#).

## Tools

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 Management Portal, PowerShell, and REST APIs help you gain additional productivity.

## Resource limitations

Azure SQL Database manages the resources available to a database using two different mechanisms: Resources governance and enforcement of limits. For more information, see:

- [DTU-based resource model limits - Single Database](#)
- [DTU-based resource model limits - Elastic pools](#)
- [vCore-based resource limits - Single Databases](#)
- [vCore-based resource limits - Elastic pools](#)

## Security

Azure SQL Database provides resources for limiting access, protecting data, and monitoring activities on a SQL Database.

- More information: [Securing your SQL Database](#).

## Authentication

- Azure SQL Database supports both SQL Server authentication users and logins, as well as [Azure Active Directory authentication](#) users and logins.
- You need to specify a particular database, instead of defaulting to the *master* database.
- You cannot use the Transact-SQL **USE myDatabaseName;** statement on SQL Database to switch to another database.
- More information: [SQL Database security: Manage database access and login security](#).

## Resiliency

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.

- Code samples: For code samples that illustrate retry logic, see samples for the language of your choice at: [Connection libraries for SQL Database and SQL Server](#).
- More information: [Error messages for SQL Database client programs](#).

## Managing 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.

## 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](#).

## Data sharding with elastic scale

Elastic scale simplifies the process of scaling out (and in).

- [Design Patterns for Multi-tenant SaaS Applications with Azure SQL Database](#).
- [Data dependent routing](#).
- [Get Started with Azure SQL Database Elastic Scale Preview](#).

## Next steps

Explore all the [capabilities of SQL Database](#).

# Getting started with JSON features in Azure SQL Database

9/25/2018 • 6 minutes to read • [Edit Online](#)

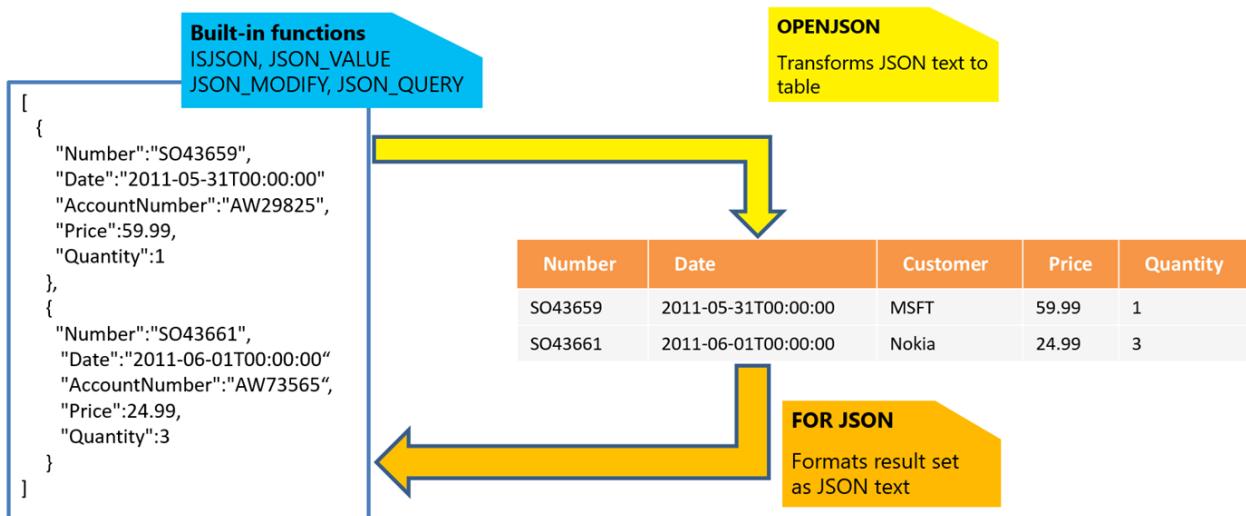
Azure SQL Database lets you parse and query data represented in JavaScript Object Notation ([JSON](#)) format, and export your relational data as JSON text.

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 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.

## Overview

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.

## 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 Orders 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
    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](#).

# Optimize performance by using In-Memory technologies in SQL Database

9/25/2018 • 17 minutes to read • [Edit Online](#)

By using In-Memory technologies in Azure SQL Database, you can achieve performance improvements with various workloads: transactional (online transactional processing (OLTP)), analytics (online analytical processing (OLAP)), and mixed (hybrid transaction/analytical processing (HTAP)). 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](#)

In-Memory technologies are available in all databases in the Premium tier, including databases in 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.

Azure SQL Database has the following In-Memory technologies:

- *In-Memory OLTP* increases transaction 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 very fast execution of analytics queries on the OLTP database, while reducing the impact on the operational workload.
- You can also have the combination of a memory-optimized table with a columnstore index. This combination enables you to perform very 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.

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.

See the following resources for more information.

In-depth information about the technologies:

- [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](#)

A quick primer on In-Memory OLTP: [Quick Start 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](#)
- [Columnstore Index: In-Memory Analytics Videos from Ignite 2016](#)

## Storage and data size

### 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`.

### 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] (<https://msdn.microsoft.com/library/cc280449.aspx#Using> Columnstore and Columnstore Archive 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.

## Moving databases that use In-Memory technologies between pricing tiers

There are never any incompatibilities or other problems when you upgrade to a higher pricing tier, such as from Standard to Premium. The available functionality and resources only increase.

But downgrading the pricing tier can negatively impact your database. The impact is especially apparent when you downgrade from 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.

### In-Memory OLTP

*Downgrading to Basic/Standard:* In-Memory OLTP isn't supported in databases in the Standard or Basic tier. In addition, it isn't possible to move a database that has any In-Memory OLTP objects to the Standard or Basic tier.

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.

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. 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
```

*Downgrading to a lower Premium tier:* Data in memory-optimized tables must fit within the In-Memory OLTP storage that is associated with the pricing tier of the database or is available in the elastic pool. If you try to lower the pricing tier or move the database into a pool that doesn't have enough available In-Memory OLTP storage, the operation fails.

### Columnstore indexes

*Downgrading to Basic or Standard:* Columnstore indexes are supported only on the 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.

*Downgrading to a lower supported tier or level:* This downgrade succeeds if the whole database fits within the maximum database size for the target pricing tier, or within the available storage in the elastic pool. There is no specific impact from the columnstore indexes.

## 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

- [Quick Start 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 \(SSDT\)](#)

# Getting Started with Temporal Tables in Azure SQL Database

9/25/2018 • 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

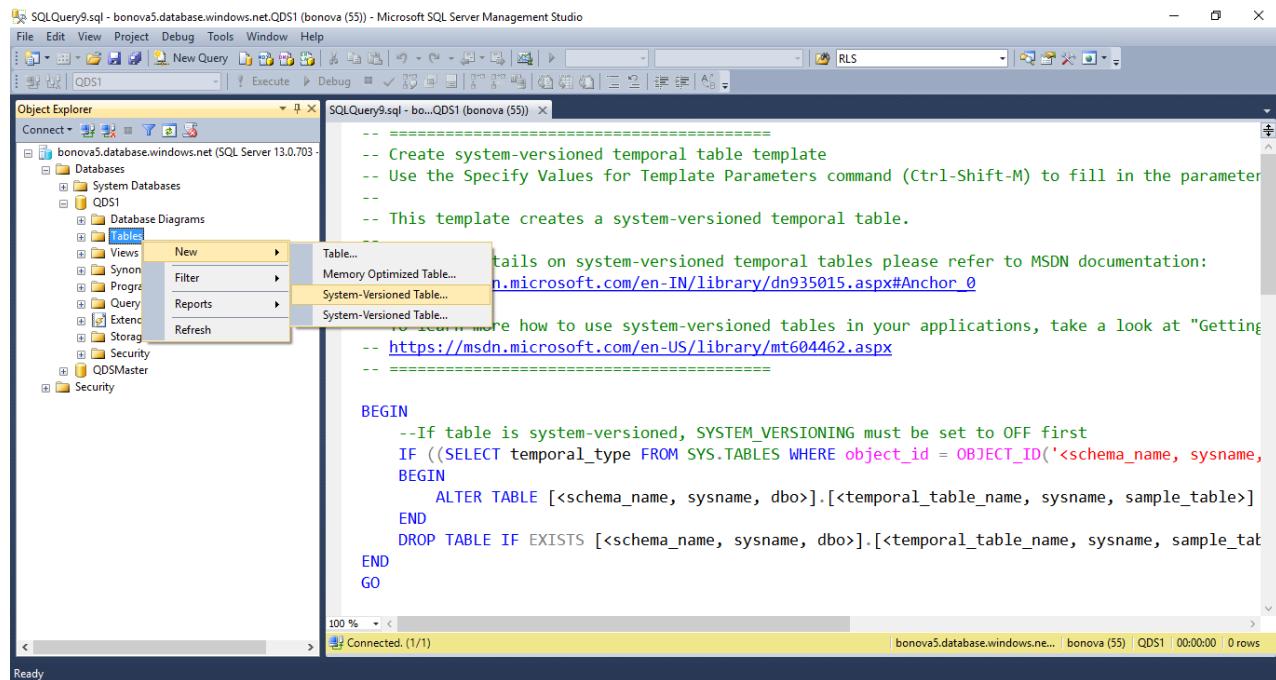
Depending on whether you are starting new development or upgrading existing application, you will either create temporal tables or modify existing ones by adding temporal attributes. In general case, your scenario can be a mix of these two options. Perform these action using [SQL Server Management Studio \(SSMS\)](#), [SQL Server Data Tools \(SSDT\)](#) or any other Transact-SQL development tool.

## IMPORTANT

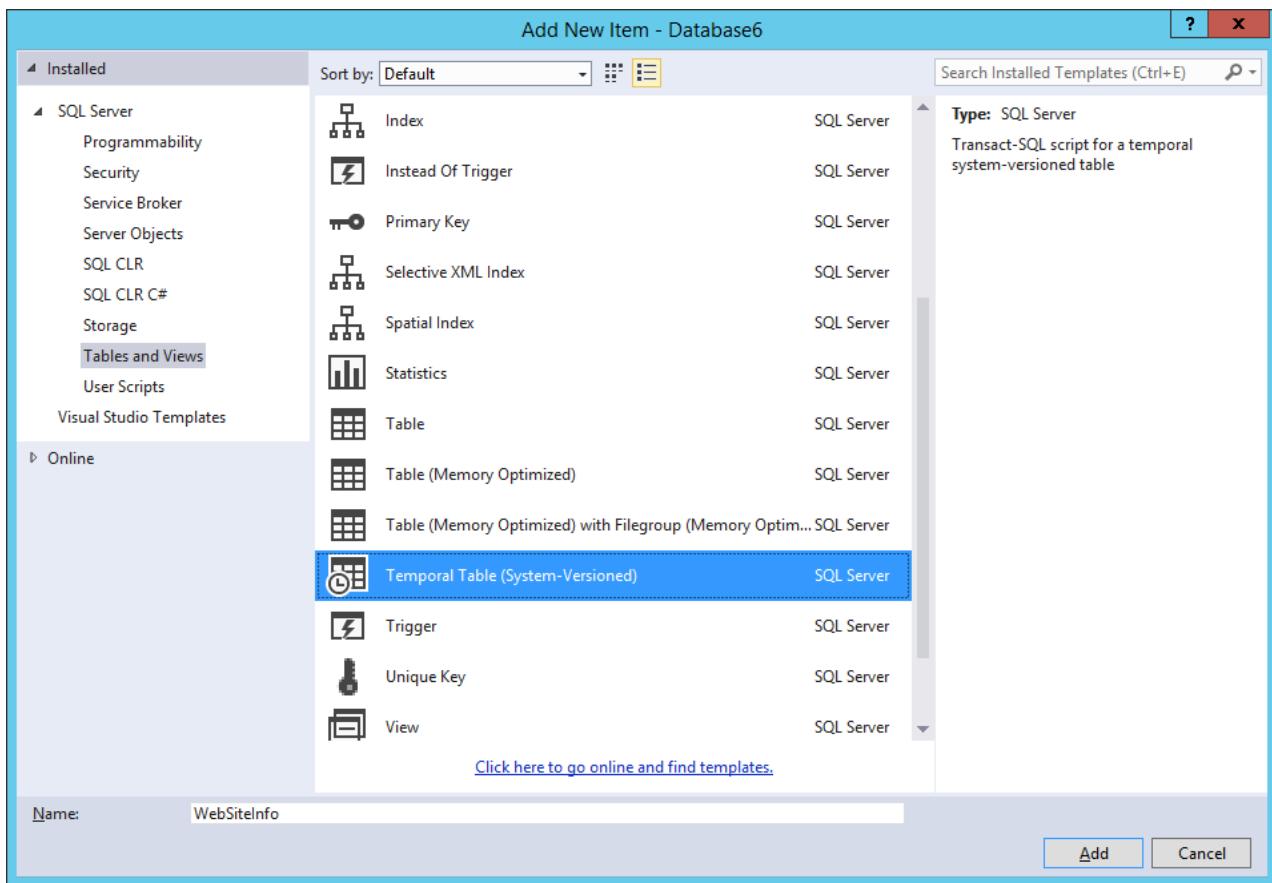
It is recommended that you always use the latest version of Management Studio to remain synchronized with updates to Microsoft Azure and SQL Database. [Update SQL Server Management Studio](#).

## 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.

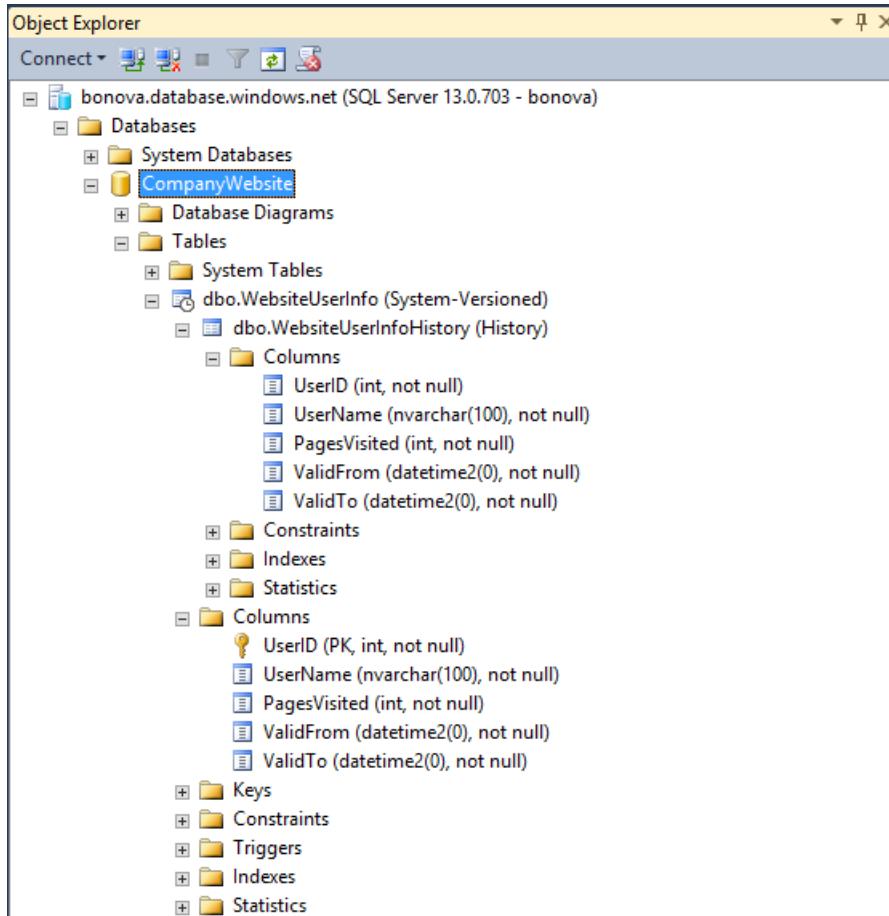
The following script shows how default index on history table can be changed to the clustered columnstore:

```

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

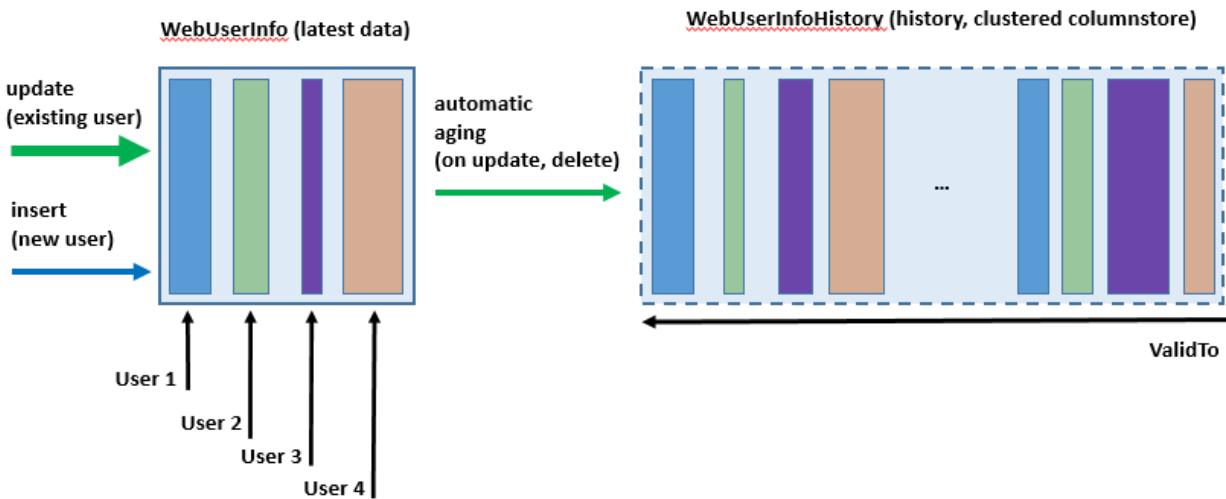
The main advantage of Temporal Tables is that you do not need to change or adjust your website in any way to

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 when user ends her/his 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
```

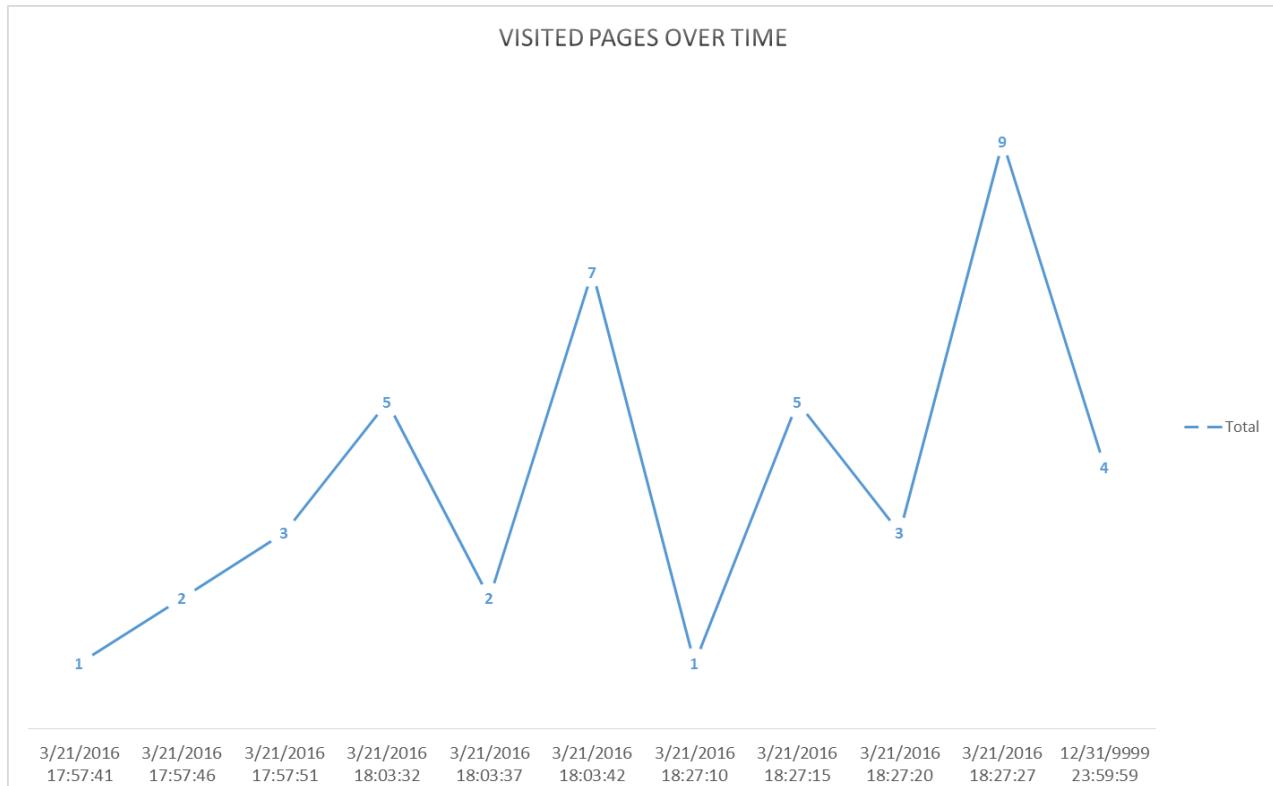
To search for activities of a specific user, within a period of time, use the **CONTAINED IN** clause:

```

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 detailed information on Temporal Tables, check out [MSDN documentation](#). Visit Channel 9 to hear a [real customer temporal implementation success story](#) and watch a [live temporal demonstration](#).

# Manage historical data in Temporal Tables with retention policy

9/25/2018 • 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:

```

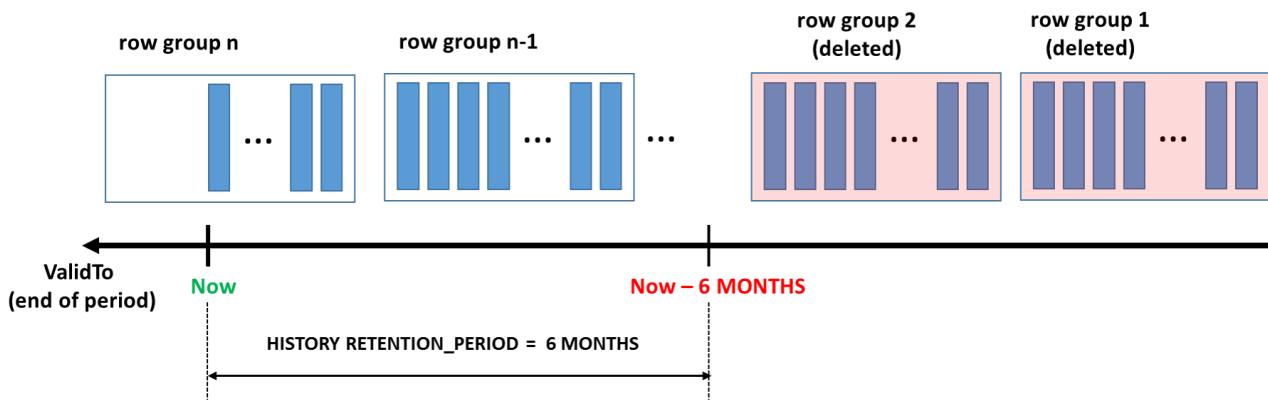
SELECT DB.is_temporal_history_retention_enabled,
SCHEMA_NAME(T1.schema_id) AS TemporalTableSchema,
T1.name as TemporalTableName, SCHEMA_NAME(T2.schema_id) AS HistoryTableSchema,
T2.name as HistoryTableName,T1.history_retention_period,
T1.history_retention_period_unit_desc
FROM sys.tables T1
OUTER APPLY (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)) \* SQLQuery2.sql - (l...UROPE\bonova (55)) \* SQLQuery3.sql - sql...stdb (BaneSa (121)) \* Query

```
SELECT * FROM WebsiteUserInfo
FOR SYSTEM_TIME ALL
```

Clustered Index Scan (Clustered)  
Scanning a clustered index, entirely or only a range.

|                                |                      |
|--------------------------------|----------------------|
| 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

Object Explorer

100 %

Results Messages Execution plan

Query 1: Query cost (relative 100)

SELECT \* FROM dbo.WebsiteUser

**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](#).

# Use In-Memory OLTP to improve your application performance in SQL Database

9/25/2018 • 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 is 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](#)

# SQL Server database migration to Azure SQL Database

10/16/2018 • 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](#).

## 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

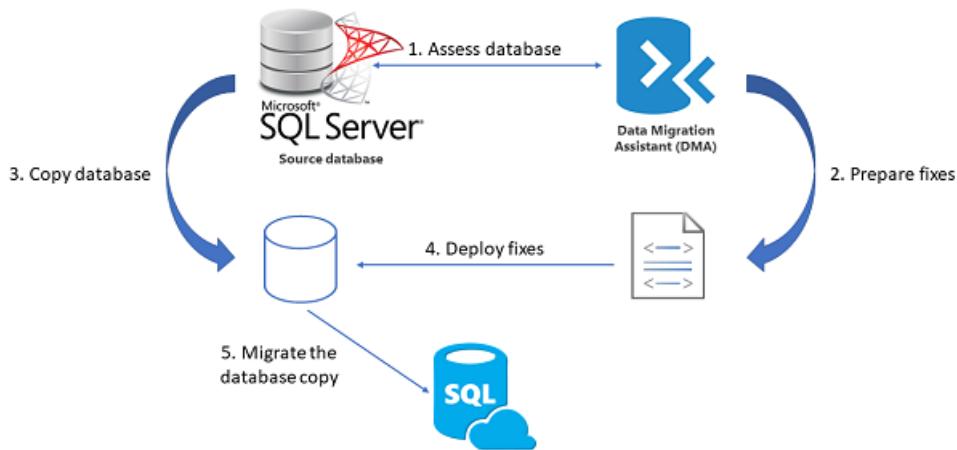
To migrate a non-SQL Server database, including Microsoft Access, Sybase, MySQL Oracle, and DB2 to Azure SQL Database, see [SQL Server Migration Assistant](#).

## 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

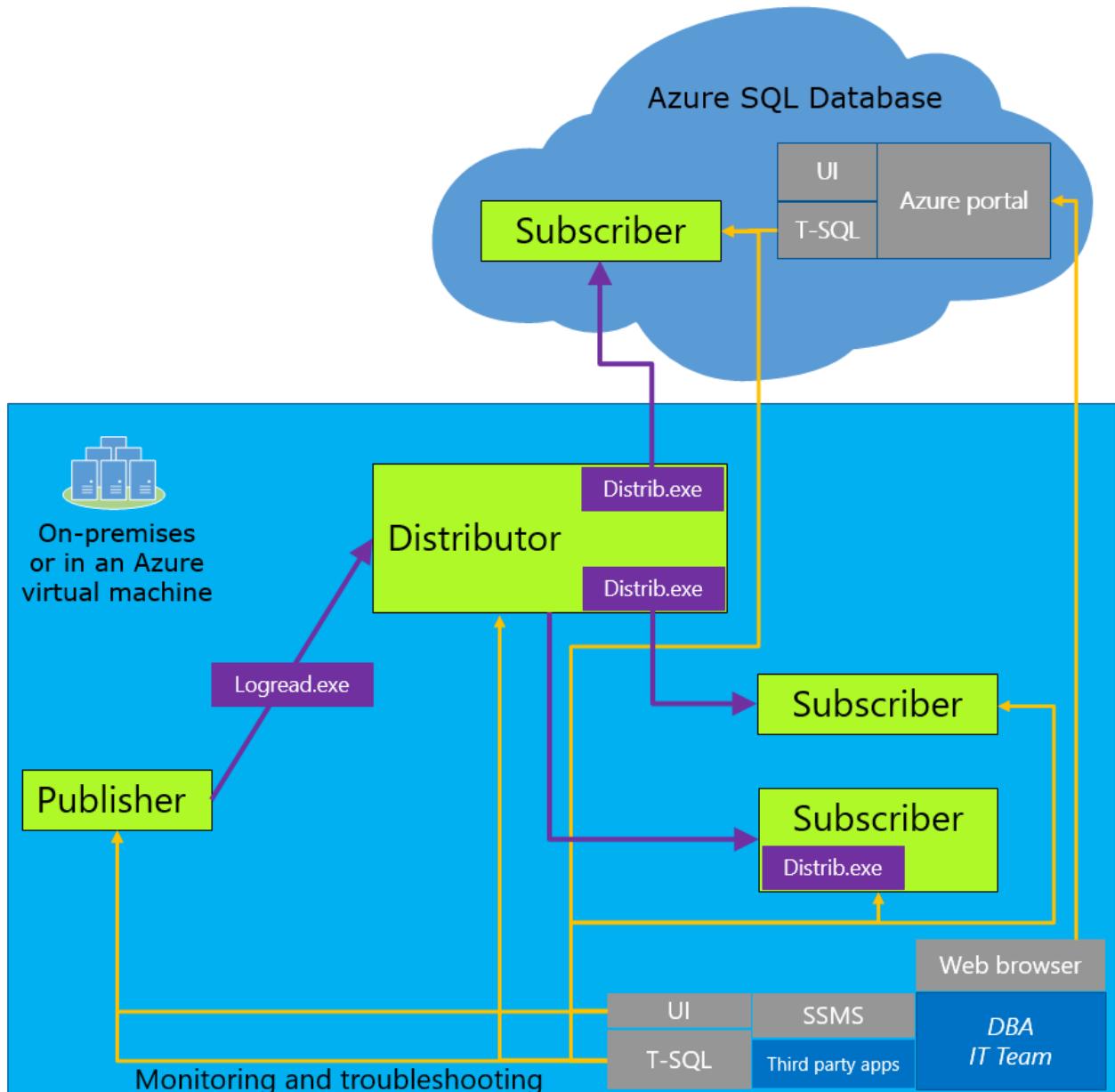
[Update statistics](#) with full scan after the migration is completed.

## 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

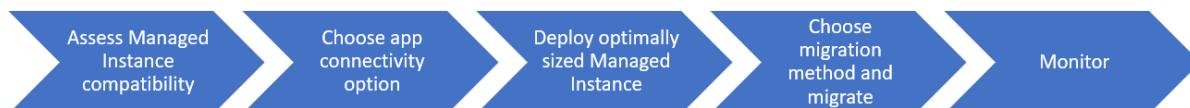
- Use the script on the Azure SQL EMEA Engineers blog to [Monitor tempdb usage during migration](#).
- Use the script on the Azure SQL EMEA Engineers blog to [Monitor the transaction log space of your database while migration is occurring](#).
- For a SQL Server Customer Advisory Team blog about migrating 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](#).

# SQL Server instance migration to Azure SQL Database Managed Instance

10/15/2018 • 7 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](#).

At a high level, the database migration process looks like:



- [Assess Managed Instance compatibility](#)
- [Choose app connectivity option](#)
- [Deploy to an optimally sized Managed Instance](#)
- [Select migration method and migrate](#)
- [Monitor applications](#)

## NOTE

To migrate a single 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. Managed Instance 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 are there some reported blocking issues that 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 in Azure SQL Database Managed Instance, you might need to consider an alternative option, such as [SQL Server on Virtual Machines in Azure](#). 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 you need to stay at a specific version of SQL Server (2012, for instance).
- If your compute requirements are much lower than Managed Instance offers in public preview (one vCore, for instance) and database consolidation is not acceptable option.

## 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 new 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.

You can select compute and storage resources at deployment time and then change it afterwards without introducing downtime for your application using the [Azure portal](#):

The screenshot shows the 'Configure Performance' step in the Azure portal for creating a SQL Managed Instance. The left sidebar lists the configuration steps: Home > SQL managed instances > SQL Managed Instance > Configure Performance. The main area is titled 'Configure Performance' and contains the following fields:

- Managed instance name:** mysqlmi
- Managed instance admin login:** (highlighted in yellow)
- Password:** (redacted)
- Confirm password:** (redacted)
- Resource group:** Create new (radio button)  Use existing (radio button)
- Location:** West US

**Performance Settings:**

- General Purpose:** For most production workloads. Options: 8 / 16 / 24 vCores, 32 GB - 8 Terabytes capacity, Fast storage.
- Storage:** In GB. Custom amount will be rounded to the nearest 32 GB value. (Slider set to 1024)
- vCores:** (Slider set to 16)

Pin to dashboard

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.

## Select migration method and migrate

Managed Instance 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 fully 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 Azure SQL Database, Managed Instance, and SQL Server in an Azure Virtual Machine. DMS is the recommended method of migration for your enterprise workloads.

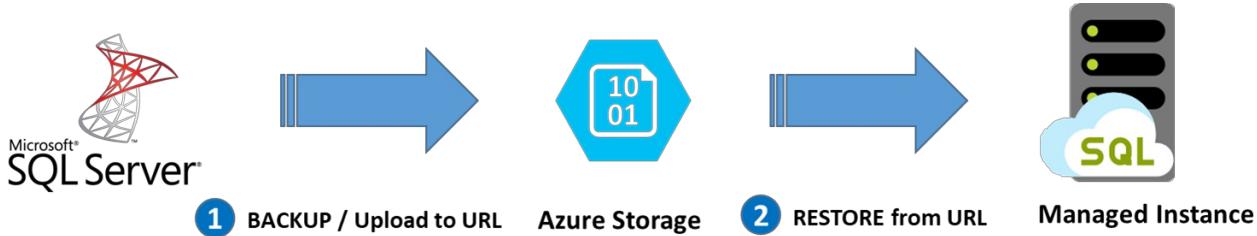
If you use SQL Server Integration Services (SSIS) on your SQL Server on premises, DMS does not yet support migrating SSIS catalog (SSISDB) that stores SSIS packages, but you can provision Azure-SSIS Integration Runtime (IR) in Azure Data Factory (ADF) that will create a new SSISDB in Azure SQL Database/Managed 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 on SQL DB Managed Instance 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 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. 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

Track application behavior and performance after migration. In Managed Instance, some changes are only enabled once the [database compatibility level has been changed](#). Database migration to Azure SQL Database keeps its original compatibility level in majority of cases. 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.

To reduce migration risks, change the database compatibility level only after performance monitoring. Use Query Store as 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 are on a fully managed platform, take advantages that are provided automatically as part of the SQL Database service. 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 \(PITR\)](#). During public preview, the retention period is fixed to seven days. Additionally, you do not need to worry about setting up high availability as [high availability](#) is built in.

To strengthen security, consider using some of the features that are available:

- Azure Active Directory Authentication at the database level
- Use [advanced security features](#) such as [Auditing](#), [Threat Detection](#), [Row-Level Security](#), and [Dynamic Data Masking](#) ) to secure your instance.

## 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](#).

# Migrate certificate of TDE protected database to Azure SQL Database Managed Instance

9/25/2018 • 3 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

Transparent Data Encryption for Azure SQL Database Managed Instance works in service-managed mode. 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, system-managed certificate.

## 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.
- AzureRM PowerShell module [installed and updated](#).
- [AzureRM.Sql module](#) version 4.10.0 or higher. Run the following commands in PowerShell to install/update the PowerShell module:

```
Install-Module -Name AzureRM.Sql  
Update-Module -Name AzureRM.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
```

|   | database_name | certificate_name |
|---|---------------|------------------|
| 1 | tempdb        | NULL             |
| 2 | TDE_DB        | TDE_Cert         |
| 3 | TDE_DB2       | TDE_Cert2        |

Query executed successfully. | 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>'
)
```

Commands completed successfully.

Query executed successfully. | 00:00:00 | 0 rows

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

1. Start with preparation steps in PowerShell:

```
# Import the module into the PowerShell session
Import-Module AzureRM

# Connect to Azure with an interactive dialog for sign-in
Connect-AzureRmAccount

# List subscriptions available and copy id of the subscription target Managed Instance belongs to
Get-AzureRmSubscription

# Set subscription for the session (replace Guid_Subscription_Id with actual subscription id)
Select-AzureRmSubscription Guid_Subscription_Id
```

2. 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
$base64EncodedCert = [System.Convert]::ToBase64String($fileContentBytes)
$securePrivateBlob = $base64EncodedCert | ConvertTo-SecureString -AsPlainText -Force
$password = "SomeStrongPassword"
$securePassword = $password | ConvertTo-SecureString -AsPlainText -Force
Add-AzureRmSqlManagedInstanceTransparentDataEncryptionCertificate -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.

# New DBA in the cloud – Managing your database in Azure SQL Database

10/8/2018 • 26 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 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 DB – 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 DB as a platform that you can readily leverage. They are the following: -

- Business continuity and disaster recovery (BCDR)
- Security and compliance
- Intelligent database monitoring and maintenance
- Data movement

## 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-centre 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 on a 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 on a geo-replicated storage subsystem 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 failover 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, Log Shipping etc. 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 logical 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 logical server. By default, all connections and databases inside the logical 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 logical server. For example, developer machine IP addresses in your organization can be added at once by specifying a range in the Firewall settings page.

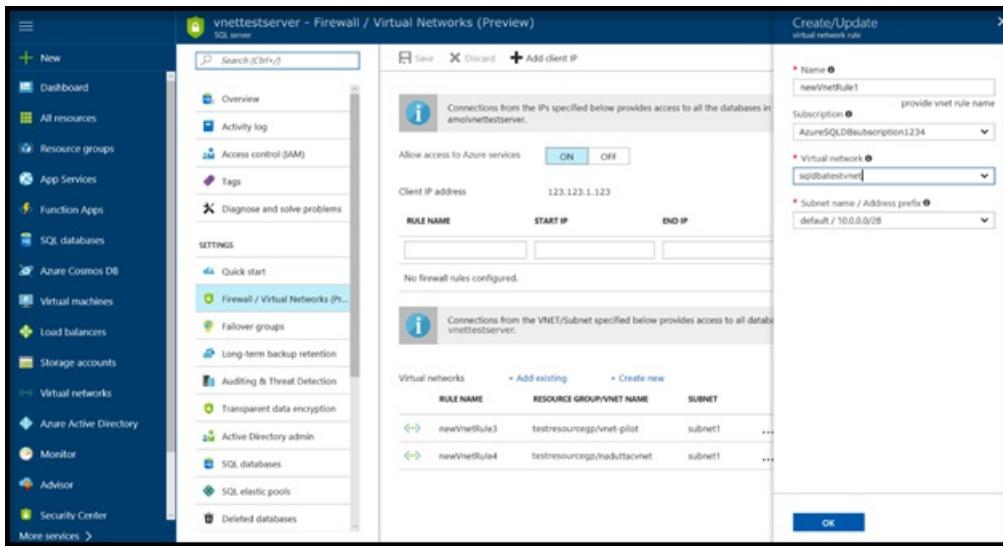
You can create firewall rules at the server level or at the database level. Server level firewall rules can either be created through the portal or through SSMS. For learning more about how to set a server and database level firewall rule, see: [Create firewall rules in SQL Database](#).

### 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 whitelist 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. You can run SSMS in Azure using [Azure RemoteApp](#). This does not require you to open outgoing connections to port 1433, the IP is static, so the DB can be open to only the RemoteApp and it supports Multi Factor Authentication (MFA).

#### 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. SQL Database 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 SQL Database 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 of his/her 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.

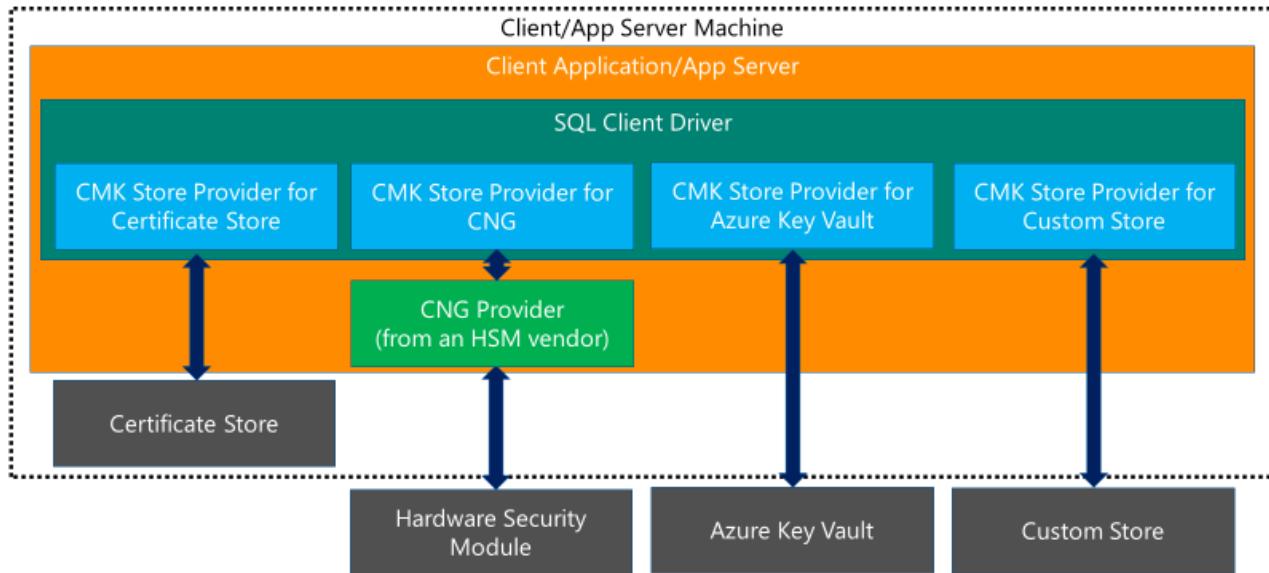
By default, the master key for Transparent Data Encryption is managed by the SQL Database service for convenience. If your organization would like control over the master key, there is an option to use Azure Key Vault] (sql-database-always-encrypted-azure-key-vault.md) as the key store. By using Azure 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 organization's compliance?

SQL Database is compliant with a range of regulatory compliances. To view the latest set of compliances that have been met, 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. [SQL 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 are provided, as well as an assessment report that can be used for helping to meet compliance.

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:

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' filter is selected. A message at the top says 'We're updating recommendations for your subscriptions. This could take some time... View details →'. Below this, it says 'Subscriptions: 2 of 7 selected' and '2 subscriptions' with a dropdown menu. The 'All types' filter is selected. A purple bar at the bottom encourages using Azure Cost Management.

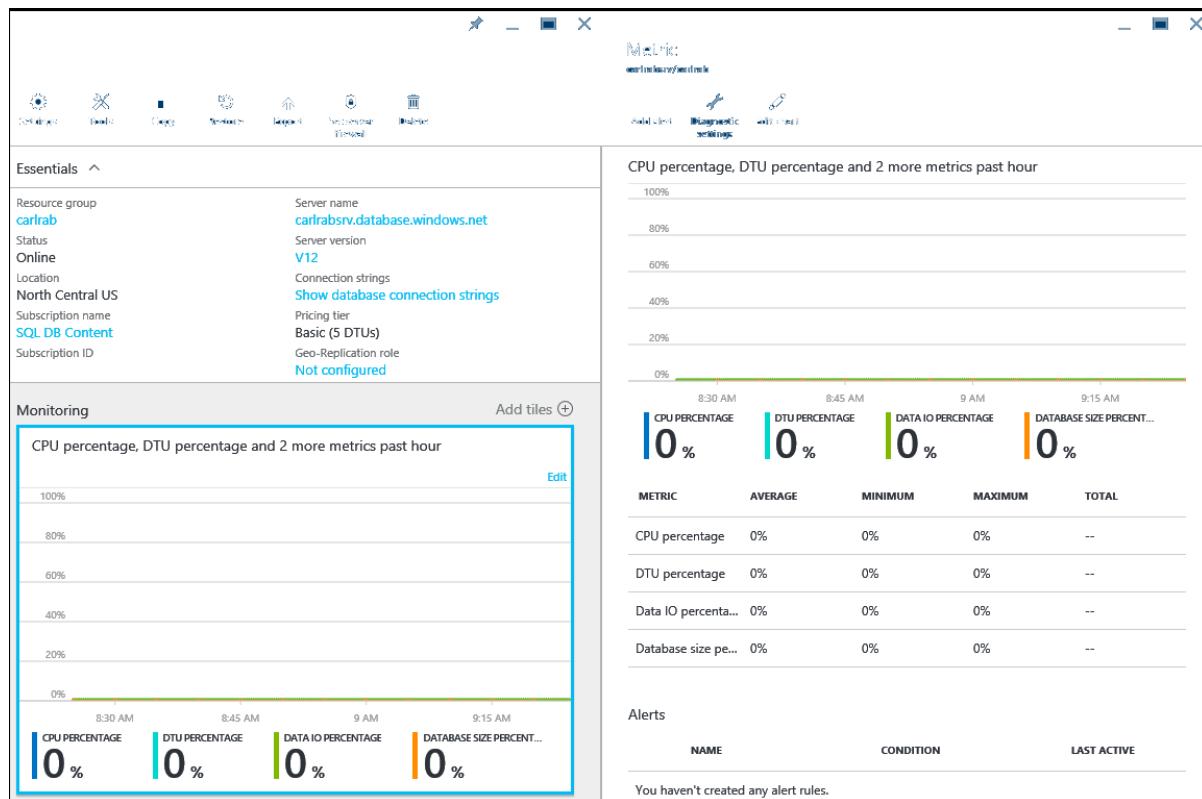
| Total recommendations | Recommendations by impact   | Impacted Resources |
|-----------------------|-----------------------------|--------------------|
| 1                     | High 1<br>Medium 0<br>Low 0 | 3                  |

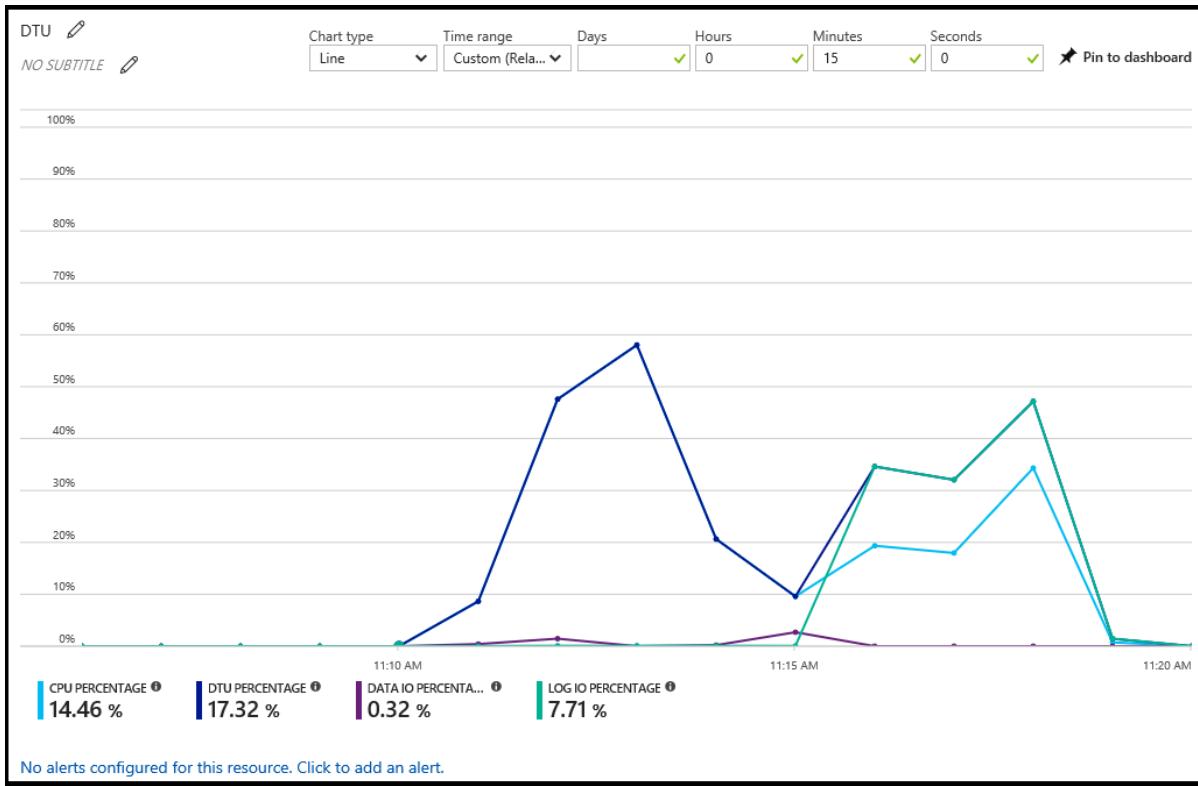
| IMPACT | DESCRIPTION                    | POTENTIAL MONITORING |
|--------|--------------------------------|----------------------|
| High   | Use SQL elastic database pools |                      |

## 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:

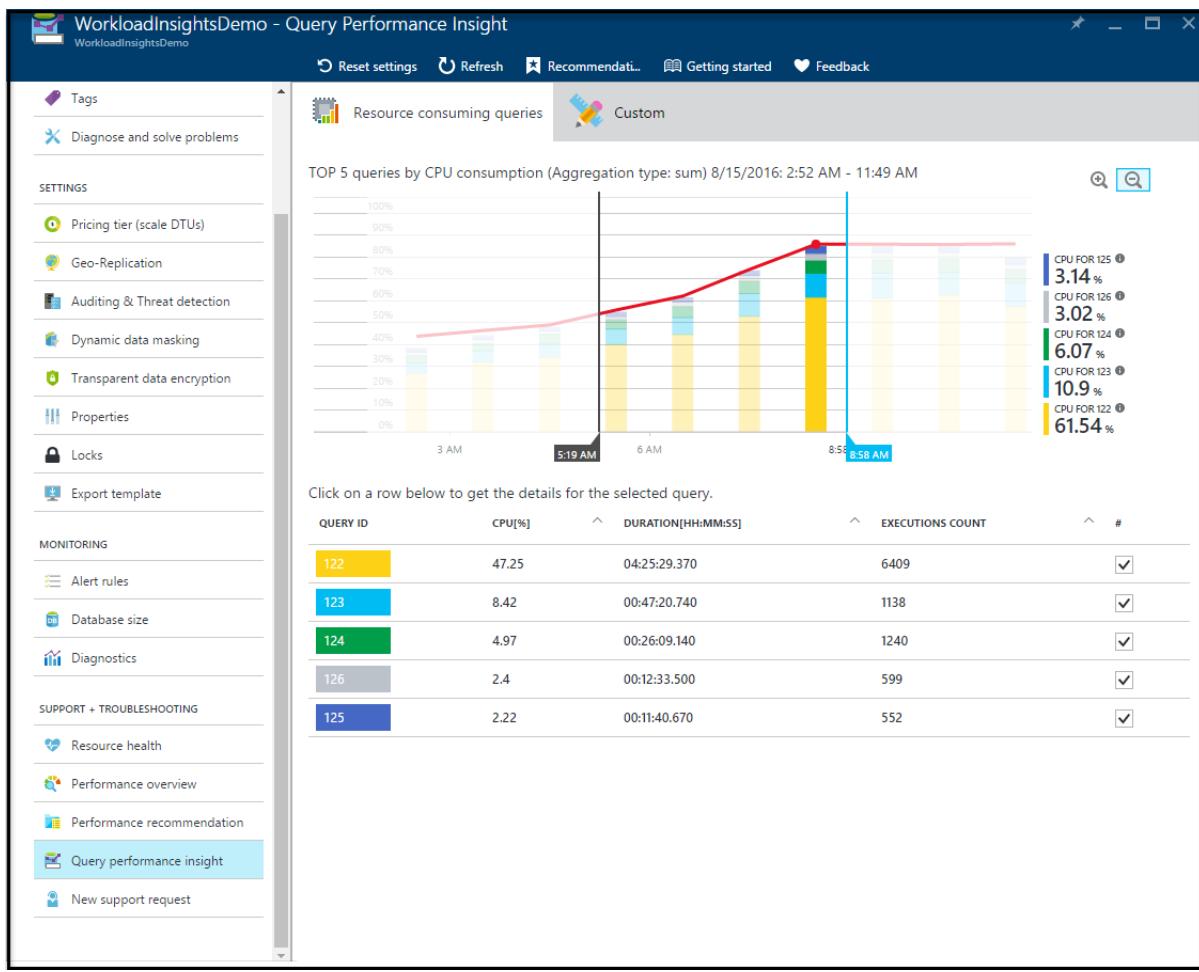
- Azure portal:** The Azure portal shows a single database's utilization by selecting the database and clicking the chart in the Overview pane. You can modify the chart to show multiple metrics, including CPU percentage, DTU percentage, Data IO percentage, Sessions percentage, and Database size percentage.





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 Log Analytics:** [Azure Log Analytics](#) allows you to collect and visualize key Azure SQL Azure 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](#).

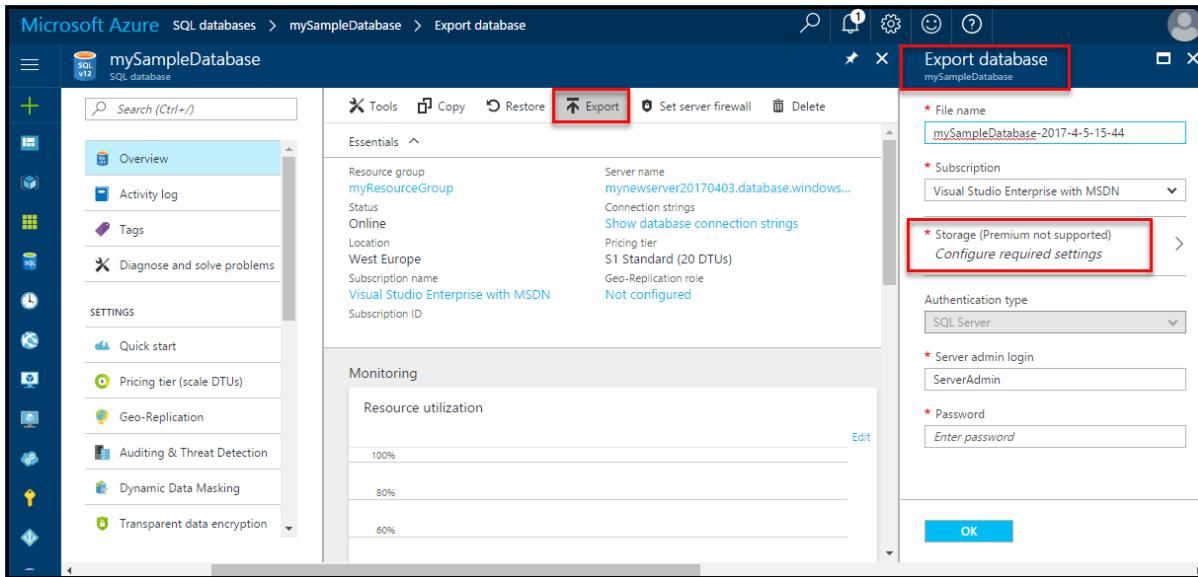
### **How often do I need to run database integrity checks for my database?**

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](#)

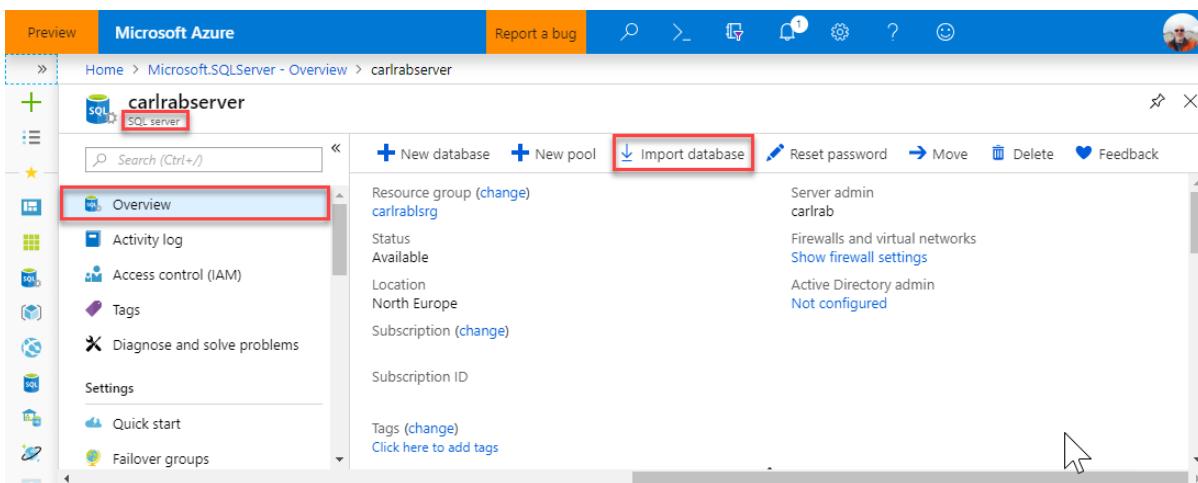
# Data movement after migration

## 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



- **Import:** You can also import data as a BACPAC file into the database using the Azure portal.



## 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](#).

# Copy an transactionally consistent copy of an Azure SQL database

10/8/2018 • 5 minutes to read • [Edit Online](#)

Azure SQL Database provides several methods for creating a transactionally consistent copy of an existing Azure SQL 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, its service tier and compute size, or 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.

## Logins in the database copy

When you copy a database to the same logical 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. All database users, their permissions, and their security identifiers (SIDs) are copied to the database copy.

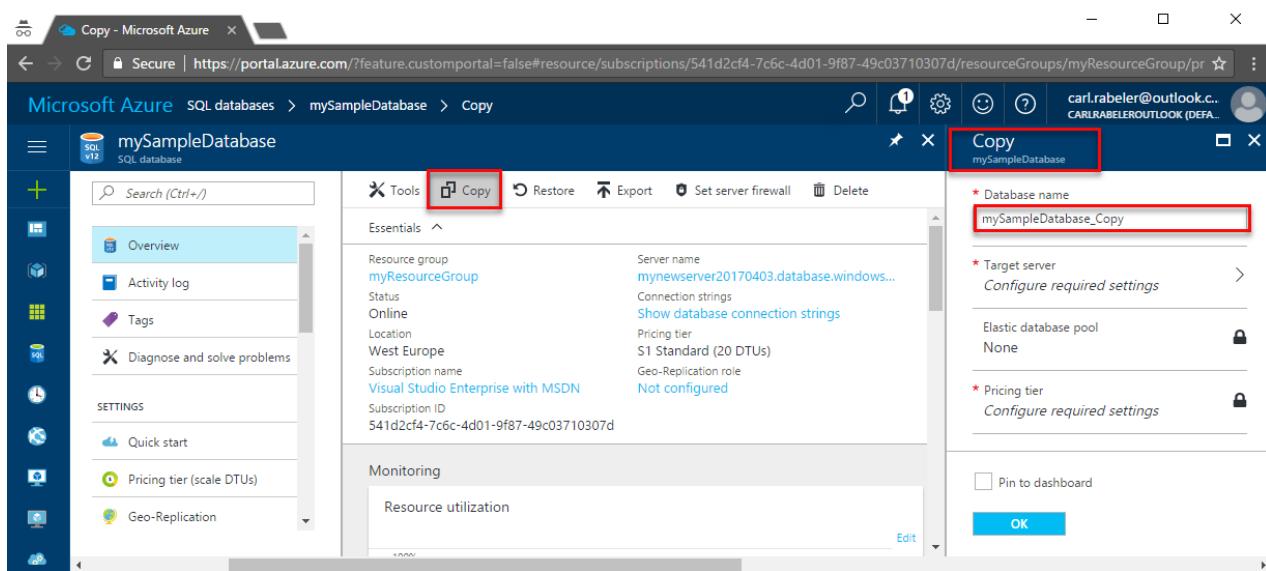
When you copy a database to a different logical server, the security principal on the new server becomes the database owner on the new database. If you use [contained database users](#) for data access, ensure that both the primary and secondary databases always have the same user credentials, so that after the copy is complete you can immediately access it with the same credentials.

If you use [Azure Active Directory](#), you can completely eliminate the need for managing credentials in the copy. However, when you copy the database to a new server, the login-based access might not work, because the logins do not exist on the new server. To learn about managing logins when you copy a database to a different logical server, see [How to manage Azure SQL database security after disaster recovery](#).

After the copying succeeds and before other users are remapped, only the login that initiated the copying, the database owner, can log in to the new database. To resolve logins 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

To copy a database by using PowerShell, use the [New-AzureRmSqlDatabaseCopy](#) cmdlet.

```
New-AzureRmSqlDatabaseCopy -ResourceGroupName "myResourceGroup" `  
-ServerName $sourceserver `  
-DatabaseName "MySampleDatabase" `  
-CopyResourceGroupName "myResourceGroup" `  
-CopyServerName $targetserver `  
-CopyDatabaseName "CopyOfMySampleDatabase"
```

For a complete sample script, see [Copy a database to a new server](#).

## Copy a database by using Transact-SQL

Log in to the master database with the server-level principal login or the login that created the database you want to copy. For database copying to succeed, logins that are not the server-level principal 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](#) statement. Executing this statement initiates the database copying process. Because copying a database is an asynchronous process, the CREATE DATABASE statement returns before the database copying is complete.

### Copy a SQL database to the same server

Log in to the master database with the server-level principal login or the login that created the database you want to copy. For database copying to succeed, logins that are not the server-level principal 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.  
-- 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 destination server, the SQL database 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 SQL database server. The login on the destination server must also be a member of the dbmanager role or be the server-level principal 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)
-- Start copying from Server1 to Server2
CREATE DATABASE Database2 AS COPY OF server1.Database1;
```

## Monitor the progress of the copying operation

Monitor the copying process by querying the sys.databases and sys.dm\_database\_copies 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. Alternatively, executing the [DROP DATABASE](#) statement on the source database also cancels the copying process.

## Resolve logins

After the new database is online on the destination server, use the [ALTER USER](#) statement to remap the users from the new database to logins on the destination 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 and is assigned a new security identifier (SID). After the copying succeeds and before other users are remapped, only the login that initiated the copying, 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 logical server, see [How to manage Azure SQL database security after disaster recovery](#).

## 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](#).

# Import a BACPAC file to a new Azure SQL Database

10/19/2018 • 4 minutes to read • [Edit Online](#)

When you need to import a database from an archive or when migrating from another platform, you can import the database schema and data from 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 imported from Azure blob storage (standard storage only) or from local storage in an on-premises location. To maximize the import speed, we recommend that you specify a higher service tier and compute size, such as a P6, and then scale to down as appropriate after the import is successful. Also, the database compatibility level after the import is based on the compatibility level of the source database.

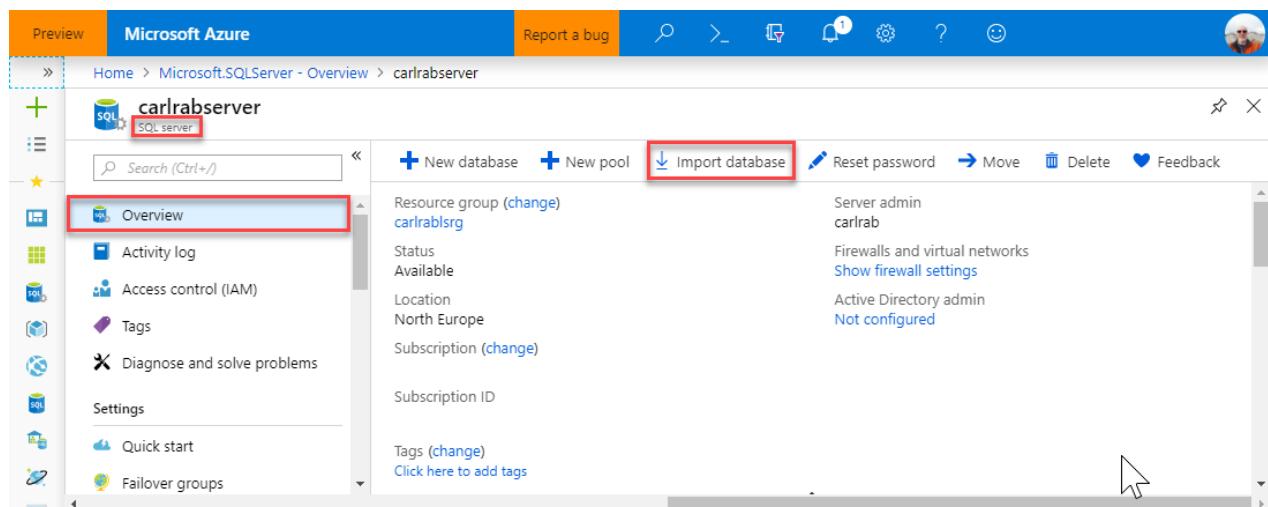
## IMPORTANT

After you migrate your database to Azure SQL 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.

## Import from a BACPAC file using Azure portal

This article provides directions for creating an Azure SQL database from a BACPAC file stored in Azure blob storage using the [Azure portal](#). Import using the Azure portal only supports importing a BACPAC file from Azure blob storage.

To import a database using the Azure portal, open the page for the server (not the page for the database) to associate the database to and then click **Import** on the toolbar. Specify the storage account and container and select the BACPAC file you want to import. Select the size of the new database (usually the same as origin) and provide the destination SQL Server credentials.



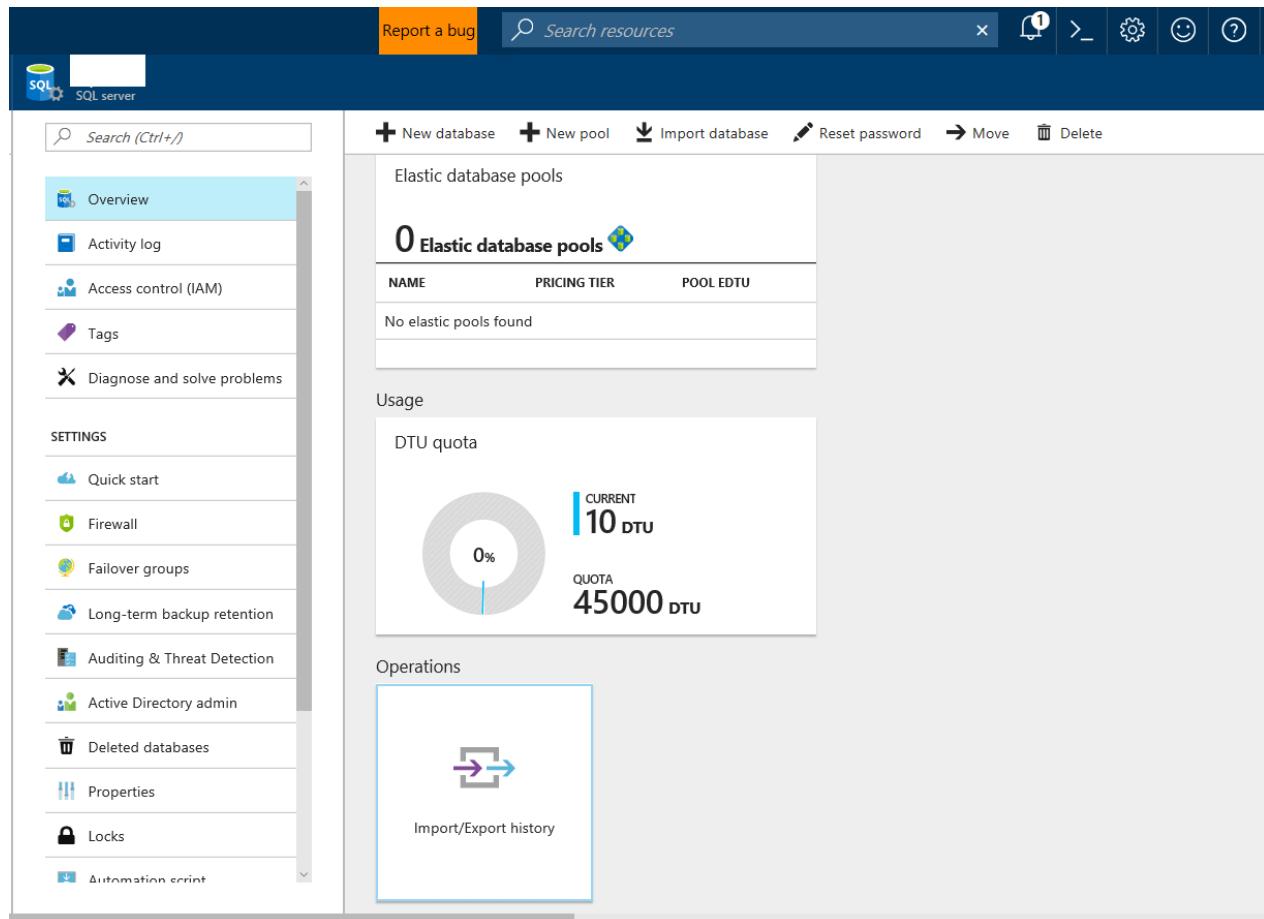
To monitor the progress of the import operation, open the page for the logical server containing the database being imported. Scroll down to **Operations** and then click **Import/Export history**.

## NOTE

Azure SQL Database Managed Instance supported importing from a BACPAC file using the other methods in this article but does not currently support migrating using the Azure portal.

## Monitor the progress of an import operation

To monitor the progress of the import operation, open the page for the logical server into which the database is being imported. Scroll down to **Operations** and then click **Import/Export history**.



The screenshot shows the Azure SQL Server management portal. The left sidebar contains navigation links such as Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Quick start, Firewall, Failover groups, Long-term backup retention, Auditing & Threat Detection, Active Directory admin, Deleted databases, Properties, Locks, Automation script), and Import/Export history. The main content area displays the 'Elastic database pools' section with a table showing 0 elastic database pools. Below it is a 'Usage' section showing a DTU quota of 10 DTU current and 45000 DTU quota. At the bottom is an 'Operations' section with a link to 'Import/Export history'.

| Import/Export history |                          |           |                      | Properties |
|-----------------------|--------------------------|-----------|----------------------|------------|
|                       |                          |           |                      |            |
| Import/Export history |                          |           |                      |            |
| OPERATION             | DATABASE                 | STATUS    | COMPLETION TIME      |            |
| Import                | carlpaasdb-restore fr... | Completed | 4/11/2016 3:21:52 PM |            |
| Export                | carlpaasdb               | Completed | 4/6/2016 3:54:00 PM  |            |

To verify the database is live on the server, click **SQL databases** and verify the new database is **Online**.

## Import from a BACPAC file using SQLPackage

To import a SQL database using the [SqlPackage](#) command-line utility, see [Import 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](#).

See the following SQLPackage command for a script example for how to import the **AdventureWorks2008R2** database from local storage to an Azure SQL Database logical server, called **mynewserver20170403** in this example. This script shows the creation of a new database called **myMigratedDatabase**, with a service tier of **Premium**, and a Service Objective of **P6**. Change these values as appropriate to your environment.

```
SqlPackage.exe /a:import /tcs:"Data Source=mynewserver20170403.database.windows.net;Initial Catalog=myMigratedDatabase;User Id=ServerAdmin;Password=<change_to_your_password>" /sf:AdventureWorks2008R2.bacpac /p:DatabaseEdition=Premium /p:DatabaseServiceObjective=P6
```

### IMPORTANT

An Azure SQL Database logical server listens on port 1433. If you are attempting to connect to an Azure SQL Database logical server from within a corporate firewall, this port must be open in the corporate firewall for you to successfully connect.

This example shows how to import a database using SqlPackage.exe 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"
```

## Import from a BACPAC file using PowerShell

Use the [New-AzureRmSqlDatabaseImport](#) cmdlet to submit an import database request to the Azure SQL Database service. Depending on the size of your database, the import operation may take some time to complete.

```
$importRequest = New-AzureRmSqlDatabaseImport -ResourceGroupName "myResourceGroup" `  
-ServerName $servername `  
-DatabaseName "MyImportSample" `  
-DatabaseMaxSizeBytes "262144000" `  
-StorageKeyType "StorageAccessKey" `  
-StorageKey $(Get-AzureRmStorageAccountKey -ResourceGroupName "myResourceGroup" -StorageAccountName  
$storageaccountname).Value[0] `  
-StorageUri "http://$storageaccountname.blob.core.windows.net/importsample/sample.bacpac" `  
-Edition "Standard" `  
-ServiceObjectiveName "P6" `  
-AdministratorLogin "ServerAdmin" `  
-AdministratorLoginPassword $(ConvertTo-SecureString -String "ASecureP@ssw0rd" -AsPlainText -Force)
```

To check the status of the import request, use the [Get-AzureRmSqlDatabaseImportExportStatus](#) cmdlet. Running this immediately after the request usually returns **Status: InProgress**. When you see **Status: Succeeded** the import is complete.

```
$importStatus = Get-AzureRmSqlDatabaseImportExportStatus -OperationStatusLink  
$importRequest.OperationStatusLink  
[Console]::Write("Importing")  
while ($importStatus.Status -eq "InProgress")  
{  
    $importStatus = Get-AzureRmSqlDatabaseImportExportStatus -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

- Import to a database in elastic pool is not supported. You can import data into a single database and then move the database to a pool.

## Import using other methods

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 [Connect to SQL Database with SQL Server Management Studio and perform a sample T-SQL query](#).
- 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 [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](#).

# Export an Azure SQL database to a BACPAC file

10/16/2018 • 4 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.

## IMPORTANT

Azure SQL Database Automated Export was retired on March 1, 2017. You can use [long-term backup retention](#) or [Azure Automation](#) to periodically archive SQL databases using PowerShell according to a schedule of your choice. For a sample, download the [sample PowerShell script](#) from Github.

## Considerations when exporting an Azure SQL database

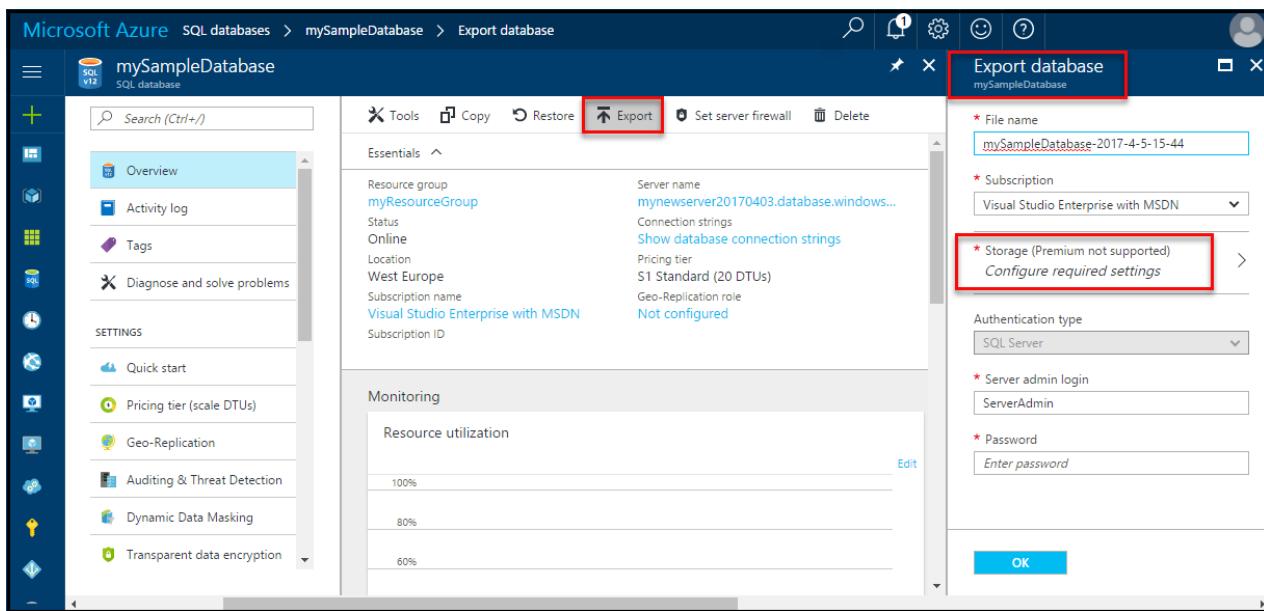
- For an export to be transactionally consistent, you must ensure either that no write activity is occurring during the export, or that you are exporting from a [transactionally consistent copy](#) of your Azure SQL database.
- If you are exporting to blob storage, the maximum size of a BACPAC file is 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.
- 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

To export a database using the [Azure portal](#), open the page for your database and click **Export** on the toolbar. Specify the BACPAC filename, provide the Azure storage account and container for the export, and provide the credentials to connect to the source database.



To monitor the progress of the export operation, open the page for the logical server containing the database being exported. Scroll down to **Operations** and then click **Import/Export history**.

The screenshot shows the 'Operations' section of the 'mySampleDatabase' page. It includes a summary of 'Elastic database pools' (0 found), a 'Usage' section showing a DTU quota of 10 DTU (Current) and 45000 DTU (Quota), and a 'Import/Export history' section which is currently empty.

| Import/Export history |                          |           |                      | Properties                     |
|-----------------------|--------------------------|-----------|----------------------|--------------------------------|
|                       |                          |           |                      | Import/Export history          |
|                       |                          |           |                      | DATABASE                       |
| OPERATION             | DATABASE                 | STATUS    | COMPLETION TIME      | carlpaasdb-restore from BACPAC |
| Import                | carlpaasdb-restore fr... | Completed | 4/11/2016 3:21:52 PM |                                |
| Export                | carlpaasdb               | Completed | 4/6/2016 3:54:00 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 SQL Package 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 also provide 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

Use the [New-AzureRmSqlDatabaseExport](#) 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-AzureRmSqlDatabaseExport -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-AzureRmSqlDatabaseImportExportStatus](#) cmdlet. Running

this immediately after the request usually returns **Status: InProgress**. When you see **Status: Succeeded** the export is complete.

```
$exportStatus = Get-AzureRmSqlDatabaseImportExportStatus -OperationStatusLink  
$exportRequest.OperationStatusLink  
[Console]::Write("Exporting")  
while ($exportStatus.Status -eq "InProgress")  
{  
    Start-Sleep -s 10  
    $exportStatus = Get-AzureRmSqlDatabaseImportExportStatus -OperationStatusLink  
    $exportRequest.OperationStatusLink  
    [Console]::Write(".")  
}  
[Console]::WriteLine("")  
$exportStatus
```

## Next steps

- To learn about long-term backup retention of an Azure SQL database backup as an alternative to exported a database for archive purposes, see [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](#) and [Migrate your first database](#).
- 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](#).

# Sync data across multiple cloud and on-premises databases with SQL Data Sync

10/16/2018 • 9 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.

## Architecture 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

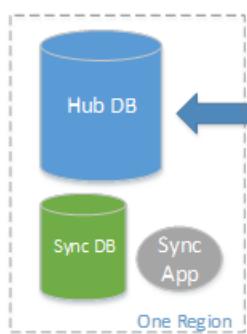
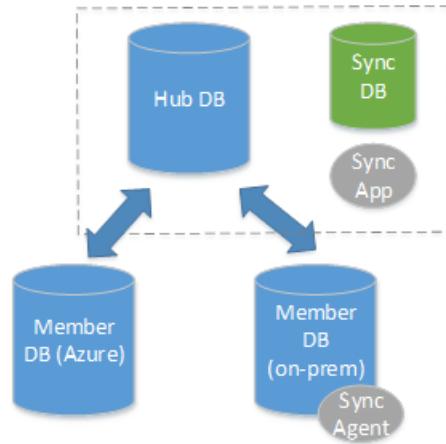


Figure 2. Sync Between 3 or more Databases



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*.

## When to use Data Sync

Data Sync is useful in cases where data needs to be kept up-to-date 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. 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 is not 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>   |
|   |  |

## 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.
- **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.

## Get started with SQL Data Sync

### Set up Data Sync in the Azure portal

- [Set up 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 is not guaranteed. Microsoft guarantees that all changes are made eventually and that Data Sync does not 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.
- Snapshot isolation must be enabled. For more info, see [Snapshot Isolation in SQL Server](#).

### General limitations

- A table cannot have an identity column that is not the primary key.
- A primary key cannot 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) cannot contain the printable characters period (.), left square bracket ([), or right square bracket (]).
- Azure Active Directory authentication is not supported.
- Tables with same name but different schema (for example, dbo.customers and sales.customers) are not 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                  |                             |
| 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 is no charge for the SQL Data Sync service itself. However, you still accrue 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)?

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 is not recommended to use SQL Data Sync to create a backup of your data. You cannot 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 does not 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 are not 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 cannot 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 are not 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 performing as expected? To monitor activity and troubleshoot issues, see the following articles:

- [Monitor Azure SQL Data Sync with Log Analytics](#)

- 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](#)

# Automate the replication of schema changes in Azure SQL Data Sync

9/25/2018 • 7 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.

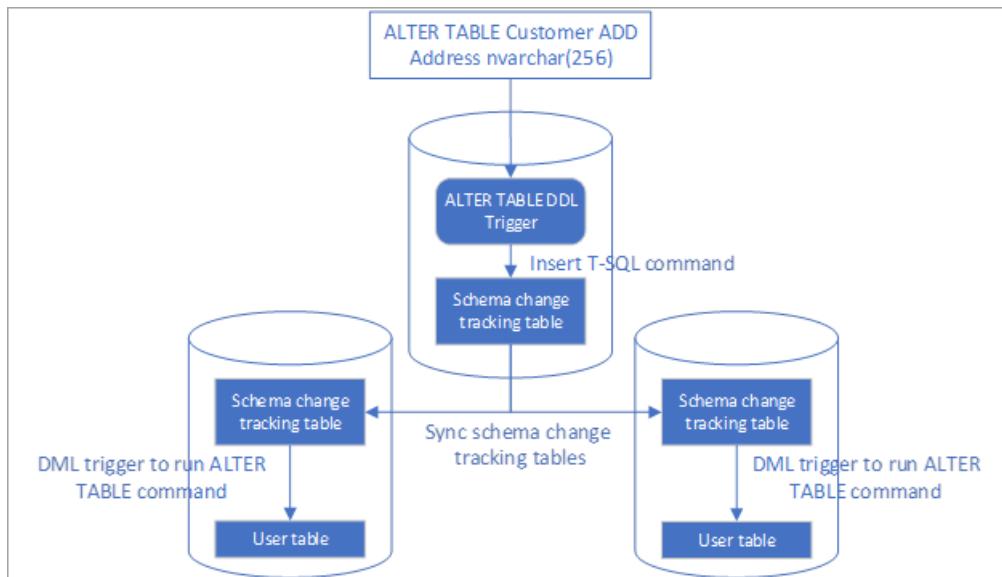
This article introduces a solution to automatically replicate schema changes to all SQL Data Sync endpoints.

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.

# Monitor SQL Data Sync with Log Analytics

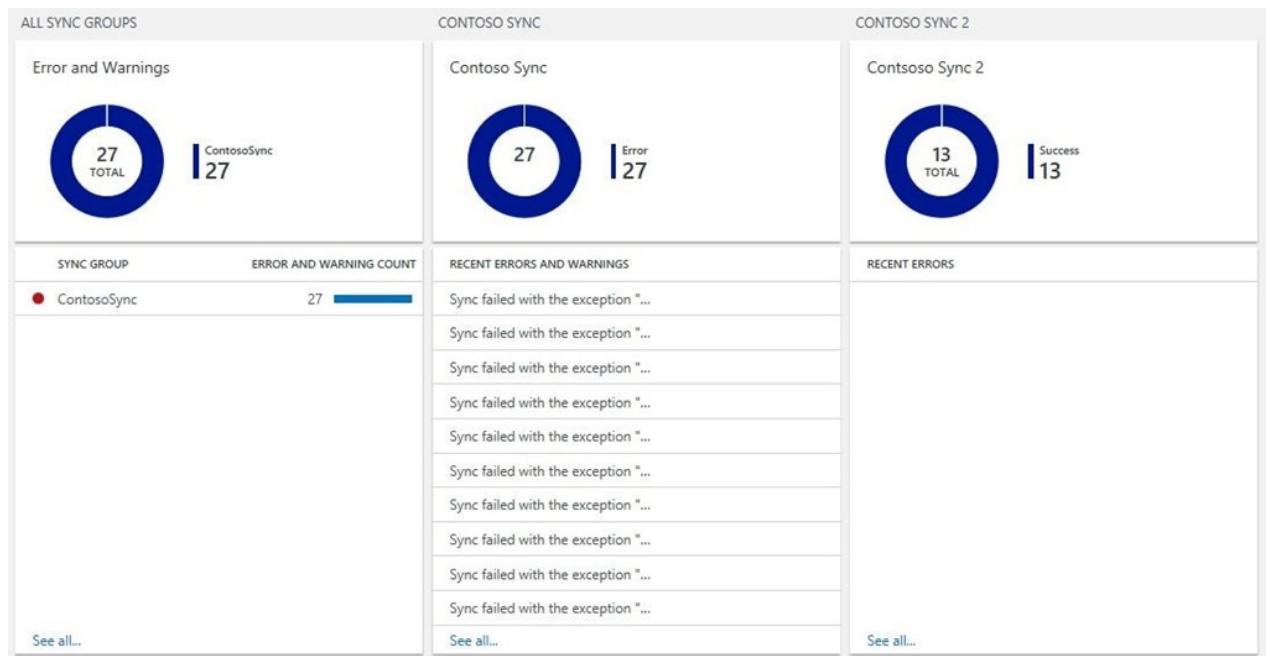
10/18/2018 • 6 minutes to read • [Edit Online](#)

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.

For an overview of SQL Data Sync, see [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#).

# 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 Log Analytics 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 [Log Analytics](#), 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  | 12 result(s)                |

# How do you set up these monitoring features?

Implement a custom Log Analytics 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 Log Analytics.
- A Log Analytics alert for email notifications.
- A Log Analytics View for monitoring.

## Samples to download

Download the following two samples:

- [Data Sync Log PowerShell Runbook](#)
- [Data Sync Log Analytics 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 Log Analytics. 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. Log Analytics information. Find this information in Azure Portal | Settings | Connected Sources. For more information about sending data to Log Analytics, see [Send data to Log Analytics 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 Log Analytics.
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 a Log Analytics Reader Alert for Email Notifications

To create an alert that uses Log Analytics, do the following things. As a prerequisite, you need to have Log Analytics 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:

```
Type=DataSyncLog\CL LogLevel\_s!=Success | measure count() by SyncGroupName\_s interval 60minute
```
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 a Log Analytics View for Monitoring

This step creates a Log Analytics 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 Log Analytics view, do the following things:

1. On the Log Analytics 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](#).

**Log Analytics:** There may be a cost associated with Log Analytics 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 [Log Analytics pricing](#).

## Code samples

Download the code samples described in this article from the following locations:

- [Data Sync Log PowerShell Runbook](#)
- [Data Sync Log Analytics View](#)

## Next steps

For more info about SQL Data Sync, see:

- [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- [Set up Azure SQL Data Sync](#)
- [Best practices for Azure SQL Data Sync](#)
- [Troubleshoot issues with Azure SQL Data Sync](#)

- Complete PowerShell examples that show how to configure SQL Data Sync:
  - [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](#)

For more info about SQL Database, see:

- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Best practices for SQL Data Sync

10/22/2018 • 8 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](#).

## 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.

## Next steps

For more information about SQL Data Sync, see:

- Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync
- Set up Azure SQL Data Sync
- Monitor Azure SQL Data Sync with Log Analytics
- Troubleshoot issues with Azure SQL Data Sync
- Complete PowerShell examples that show how to configure SQL Data Sync:
  - [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](#)

For more information about SQL Database, see:

- [SQL Database overview](#)
- [Database lifecycle management](#)

# Troubleshoot issues with SQL Data Sync

9/25/2018 • 14 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](#).

## 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 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 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

- [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 AgentServiceSetup.msi file is LocalAgentHost.msi, generate and examine log files by using the following command lines:

- For installs: `msiexec.exe /i SQLDataSyncAgent-Preview-ENU.msi /l\*v LocalAgentSetup.InstallLog`
- For uninstalls: `msiexec.exe /x SQLDataSyncAgent-se-ENU.msi /l\*v LocalAgentSetup.InstallLog`

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.

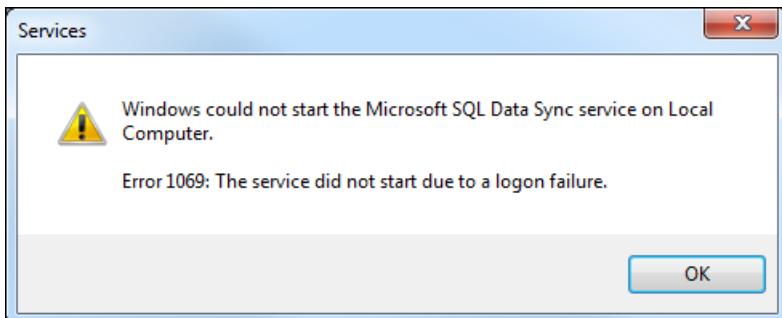
These scenarios might cause this issue:

- **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)

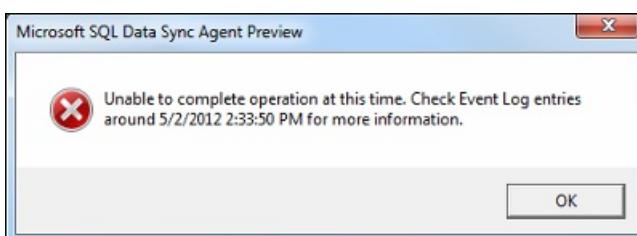
You discover that the agent isn't running on a computer that hosts SQL Server. When you attempt to manually start the agent, you see a dialog box that displays the message, "Error 1069: The service did not start due to a logon failure."



- **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.

## **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:

- [Sync data across multiple cloud and on-premises databases with Azure SQL Data Sync](#)
- [Set up Azure SQL Data Sync](#)
- [Best practices for Azure SQL Data Sync](#)
- [Monitor Azure SQL Data Sync with Log Analytics](#)
- Complete PowerShell examples that show how to configure SQL Data Sync:
  - [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](#)

For more information about SQL Database, see:

- [SQL Database Overview](#)
- [Database Lifecycle Management](#)

# Load data from CSV into Azure SQL Database (flat files)

9/25/2018 • 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 logical 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 data to or from Azure SQL Database by using Azure Data Factory

10/23/2018 • 14 minutes to read • [Edit Online](#)

This article explains how to use Copy Activity in Azure Data Factory to copy data from or to Azure SQL Database. It builds on the [Copy Activity overview](#) article, which presents a general overview of Copy Activity.

## Supported capabilities

You can copy data from or to Azure SQL Database to any supported sink data store. And you can copy data from any supported source data store to Azure SQL Database. For a list of data stores that are supported as sources or sinks by Copy Activity, see the [Supported data stores and formats](#) table.

Specifically, this Azure SQL Database connector supports these functions:

- Copy 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, retrieve data by using a SQL query or stored procedure.
- As a sink, append data to a destination table or invoke a stored procedure with custom logic during the copy.

### IMPORTANT

If you copy data by using 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 is used to connect to Azure SQL Database.

## Get started

You can use one of the following tools or SDKs to use 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 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. Mark this field as a <b>SecureString</b> to store it securely in Data Factory, or <a href="#">reference a secret stored 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 a <b>SecureString</b> to store it securely in 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 (domain name or tenant ID) under which your application resides. Retrieve it by hovering the mouse in the top-right corner of the Azure portal.   | Yes, when you use Azure AD authentication with a service principal. |
| connectVia          | The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or a self-hosted integration runtime if your data store is located in a private network. If not specified, it uses the default Azure Integration Runtime. | 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 error with error code as "UserErrorFailedToConnectToSqlServer" and 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": {
        "type": "SecureString",
        "value": "Server=tcp:<servername>.database.windows.net,1433;Database=<dbname>;User ID=<username>@<servername>;Password=<password>;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
      }
    },
    "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 SSMS, 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:

```
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": {
                "type": "SecureString",
                "value": "Server=tcp:<servername>.database.windows.net,1433;Database=<dbname>;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 service 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 MSI-based Azure AD application token authentication, follow these steps:

1. **Create a group in Azure AD.** Make the factory MSI a member of the group.
  - a. Find the data factory service identity from the Azure portal. Go to your data factory's **Properties**. Copy the SERVICE IDENTITY ID.
  - b. Install the [Azure AD PowerShell](#) module. Sign in by using the `Connect-AzureAD` command. Run the following commands to create a group and add the data factory MSI as a member.

```
$Group = New-AzureADGroup -DisplayName "<your group name>" -MailEnabled $false -SecurityEnabled $true -MailNickname "NotSet"
Add-AzureAdGroupMember -ObjectId $Group.ObjectId -RefObjectId "<your data factory service identity 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 can be an Azure AD user or Azure AD group. If you grant the group with MSI an admin role, skip steps 3 and 4. The administrator will have full access to the database.
3. **Create contained database users** for the Azure AD group. Connect to the database from or to which you want to copy data by using tools like SSMS, with an Azure AD identity that has at least ALTER ANY USER permission. Run the following T-SQL:

```
CREATE USER [your AAD group name] FROM EXTERNAL PROVIDER;
```

4. **Grant the Azure AD group needed permissions** as you normally do for SQL users and others. For example, run the following code:

```
EXEC sp_addrolemember [role name], [your AAD group name];
```

## 5. Configure an Azure SQL Database linked service in Azure Data Factory.

### Linked service example that uses MSI authentication

```
{
    "name": "AzureSqlDbLinkedService",
    "properties": {
        "type": "AzureSqlDatabase",
        "typeProperties": {
            "connectionString": {
                "type": "SecureString",
                "value": "Server=tcp:<servername>.database.windows.net,1433;Database=<dbname>;Connection Timeout=30"
            },
            "connectVia": {
                "referenceName": "<name of Integration Runtime>",
                "type": "IntegrationRuntimeReference"
            }
        }
    }
}
```

## Dataset properties

For a full list of sections and properties available for defining datasets, see the [Datasets](#) article. This section provides a list of properties supported by the Azure SQL Database dataset.

To copy data from or to Azure SQL Database, set the **type** property of the dataset to **AzureSqlTable**. The following properties are supported:

| PROPERTY  | DESCRIPTION   | REQUIRED |
|-----------|---|----------|
| type      | The <b>type</b> property of the dataset must be set to <b>AzureSqlTable</b> .                       | Yes      |
| tableName | The name of the table or view in the Azure SQL Database instance that the linked service refers to. | Yes      |

### Dataset properties example

```
{
    "name": "AzureSQLDbDataset",
    "properties": {
        "type": "AzureSqlTable",
        "linkedServiceName": {
            "referenceName": "<Azure SQL Database linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "MyTable"
        }
    }
}
```

# Copy Activity properties

For a full list of sections and properties available for defining activities, see the [Pipelines](#) article. 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, set the **type** property in the Copy Activity source to **SqISource**. 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>SqISource</b> .  | Yes      |
| sqlReaderQuery               | Use the custom SQL query to read data. Example:<br><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. Names and casing of parameters must match the names and casing of the stored procedure parameters. | No       |

## Points to note

- If the **sqlReaderQuery** is specified for the **SqISource**, Copy Activity runs this query against the Azure SQL Database source to get the data. Or you can specify a stored procedure. Specify **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. `select column1, column2 from mytable` runs against Azure SQL Database. If the dataset definition doesn't have the **structure**, all columns are selected from the table.
- When you use **sqlReaderStoredProcedureName**, you still need to specify a dummy **tableName** property in the dataset JSON.

## 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": "SqlSource",
                "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": "SqlSource",
                "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

To copy data to Azure SQL Database, set the **type** property in the Copy Activity sink to **SqISink**. 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>SqISink</b> .  | Yes                       |
| writeBatchSize               | Inserts data into the SQL table when the buffer size reaches <b>writeBatchSize</b> . The allowed value is <b>integer</b> (number of rows).  | No. The default is 10000. |
| writeBatchTimeout            | The wait time for the batch insert operation to finish before it times out. The allowed value is <b>timespan</b> . Example: "00:30:00" (30 minutes).  | No                        |
| preCopyScript                | Specify a SQL query for Copy Activity to run before writing data into Azure SQL Database. It's only invoked once per copy run. Use this property to clean up the preloaded data.  | No                        |
| sqlWriterStoredProcedureName | <p>The name of the stored procedure that defines how to apply source data into a target table. An example is to do upserts or transform by using your own business logic.</p> <p>This stored procedure is <b>invoked per batch</b>. For operations that only run once and have nothing to do with source data, use the <b>preCopyScript</b> property. Example operations are delete and truncate.</p> | 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                        |

| PROPERTY           | DESCRIPTION  | REQUIRED |
|--------------------|--|----------|
| sqlWriterTableType | Specify a table type name to be used in the stored procedure. Copy Activity makes the data being moved available in a temporary table with this table type. Stored procedure code can then merge the data being copied with existing data. | No       |

#### TIP

When you copy data to Azure SQL Database, Copy Activity appends data to the sink table by default. To do an upsert or additional business logic, use the stored procedure in **SqlSink**. Learn more details from [Invoking stored procedure from SQL Sink](#).

#### Append data example

```
"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": "SqlSink",
                "writeBatchSize": 100000
            }
        }
    }
]
```

#### Invoke a stored procedure during copy for upsert example

Learn more details from [Invoking stored procedure from 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": "SqlSink",
                "sqlWriterStoredProcedureName": "CopyTestStoredProcedureWithParameters",
                "sqlWriterTableType": "CopyTestTableType",
                "storedProcedureParameters": {
                    "identifier": { "value": "1", "type": "Int" },
                    "stringData": { "value": "str1" }
                }
            }
        }
    }
]

```

## Identity columns in the target database

This section shows you how to copy data from a source table without an identity column to a destination table with an identity column.

### Source table

```

create table dbo.SourceTbl
(
    name varchar(100),
    age int
)

```

### Destination table

```

create table dbo.TargetTbl
(
    identifier int identity(1,1),
    name varchar(100),
    age int
)

```

### NOTE

The target table has an identity column.

### Source dataset JSON definition

```
{
    "name": "SampleSource",
    "properties": {
        "type": "AzureSqlTable",
        "linkedServiceName": {
            "referenceName": "TestIdentitySQL",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "SourceTbl"
        }
    }
}
```

#### Destination dataset JSON definition

```
{
    "name": "SampleTarget",
    "properties": {
        "structure": [
            { "name": "name" },
            { "name": "age" }
        ],
        "type": "AzureSqlTable",
        "linkedServiceName": {
            "referenceName": "TestIdentitySQL",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "TargetTbl"
        }
    }
}
```

#### NOTE

Your source and target table have different schema.

The target has an additional column with an identity. In this scenario, you must specify the **structure** property in the target dataset definition, which doesn't include the identity column.

## Invoke stored procedure from SQL sink

When you copy data into Azure SQL Database, you can also configure and invoke a user-specified stored procedure with additional parameters.

You can use a stored procedure when built-in copy mechanisms don't serve the purpose. They're typically used when an upsert, insert plus update, or extra processing must be done before the final insertion of source data into the destination table. Some extra processing examples are merge columns, look up additional values, and insertion 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 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.

#### Output dataset

```
{
    "name": "AzureSQLDbDataset",
    "properties":
    {
        "type": "AzureSqlTable",
        "linkedServiceName": {
            "referenceName": "<Azure SQL Database linked service name>",
            "type": "LinkedServiceReference"
        },
        "typeProperties": {
            "tableName": "Marketing"
        }
    }
}
```

Define the **SqISink** section in Copy Activity:

```
"sink": {
    "type": "SqlSink",
    "SqlWriterTableType": "MarketingType",
    "SqlWriterStoredProcName": "spOverwriteMarketing",
    "storedProcParameters": {
        "category": {
            "value": "ProductA"
        }
    }
}
```

In your database, define the stored procedure with the same name as the **SqIWriterStoredProcName**. 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 should be the same as the **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
```

In your database, define the table type with the same name as the **sqlWriterTableType**. The schema of the table type should be 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,
)
```

The stored procedure feature takes advantage of [Table-Valued Parameters](#).

**NOTE**

If you write to Money/Smallmoney data type by invoking Stored Procedure, values may be rounded. Specify the corresponding data type in TVP as Decimal instead of Money/Smallmoney to mitigate.

## Data type mapping for Azure SQL Database

When you copy data from or to Azure SQL Database, the following mappings are used from Azure SQL Database data types to Azure Data Factory interim data types. See [Schema and data type mappings](#) to learn how Copy Activity maps the source schema and data type to the sink.

| AZURE SQL DATABASE DATA TYPE          | 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[]                         |

| AZURE SQL DATABASE DATA TYPE | DATA FACTORY INTERIM DATA TYPE |
|------------------------------|--------------------------------|
| smalldatetime                | DateTime                       |
| smallint                     | Int16                          |
| smallmoney                   | Decimal                        |
| sql_variant                  | Object *                       |
| text                         | String, Char[]                 |
| time                         | TimeSpan                       |
| timestamp                    | Byte[]                         |
| tinyint                      | Byte                           |
| uniqueidentifier             | Guid                           |
| varbinary                    | Byte[]                         |
| varchar                      | String, Char[]                 |
| xml                          | Xml                            |

## Next steps

For a list of data stores supported as sources and sinks by Copy Activity in Azure Data Factory, see [Supported data stores and formats](#).

# Manage file space in Azure SQL Database

10/23/2018 • 7 minutes to read • [Edit Online](#)

This article describes different types of storage space 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.

## Overview

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 DB 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.

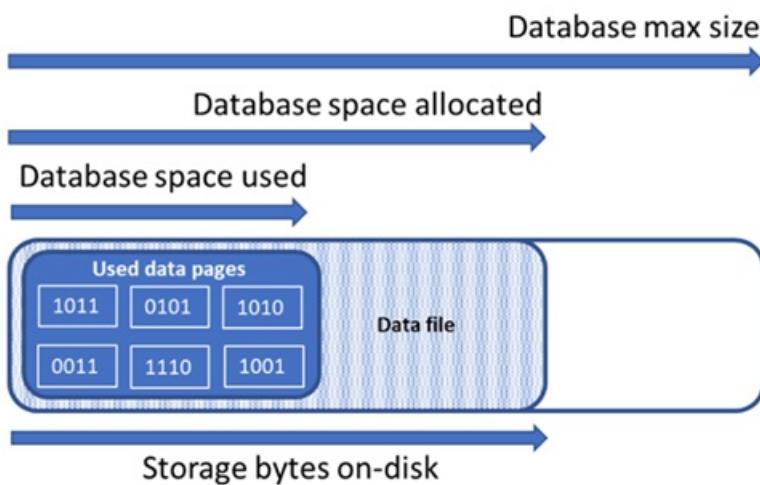
## Understanding types of storage space for a database

Understanding the following storage space quantities are important for managing the file space of a database.

| DATABASE QUANTITY | DEFINITION | COMMENTS |
|-------------------|------------|----------|
|-------------------|------------|----------|

| 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 database for storage space information

The following queries can be used to determine storage space quantities for a 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 * 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 PowerShell script 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.

The PowerShell script requires SQL Server PowerShell module – see [Download PowerShell module](#) to install.

```

# Resource group name
$resourceGroupName = "rg1"
# Server name
$serverName = "ls2"
# Elastic pool name
$poolName = "ep1"
# User name for server
$userName = "name"
# Password for server
$password = "password"

# Get list of databases in elastic pool
$databasesInPool = Get-AzureRmSqlElasticPoolDatabase `

    -ResourceGroupName $resourceGroupName `

    -ServerName $serverName `

    -ElasticPoolName $poolName

$databaseStorageMetrics = @()

# For each database in the elastic pool,
# get its 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

## 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](#).

# Connectivity libraries and frameworks for SQL Server

9/25/2018 • 2 minutes to read • [Edit Online](#)

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 | <a href="#">Microsoft ADO.NET for SQL Server</a>     | <a href="#">Download</a>   | <a href="#">Get started</a> |
| Java     | Windows, Linux, macOS | <a href="#">Microsoft JDBC driver for SQL Server</a> | <a href="#">Download</a>   | <a href="#">Get started</a> |
| PHP      | Windows, Linux, macOS | <a href="#">PHP SQL driver for SQL Server</a>        | Operating system:<br>* <a href="#">Windows</a><br>* <a href="#">Linux</a><br>* <a href="#">macOS</a> | <a href="#">Get started</a> |
| Node.js  | Windows, Linux, macOS | <a href="#">Node.js driver for SQL Server</a>        | <a href="#">Install</a>  | <a href="#">Get started</a> |
| Python   | Windows, Linux, macOS | <a href="#">Python SQL driver</a>                    | Install choices:<br>* <a href="#">pymssql</a><br>* <a href="#">pyodbc</a>                            | <a href="#">Get started</a> |
| Ruby     | Windows, Linux, macOS | <a href="#">Ruby driver for SQL Server</a>           | <a href="#">Install</a>  | <a href="#">Get started</a> |
| C++      | Windows, Linux, macOS | <a href="#">Microsoft ODBC driver for SQL Server</a> | <a href="#">Download</a>   |                             |

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 | <a href="#">Entity Framework</a><br><a href="#">Entity Framework Core</a> |
| Java     | Windows, Linux, macOS | <a href="#">Hibernate ORM</a>   |
| PHP      | Windows, Linux        | <a href="#">Laravel (Eloquent)</a>  |

| LANGUAGE | PLATFORM              | ORM(S)                        |
|----------|-----------------------|-------------------------------|
| Node.js  | Windows, Linux, macOS | <a href="#">Sequelize ORM</a> |
| Python   | Windows, Linux, macOS | <a href="#">Django</a>        |
| Ruby     | Windows, Linux, macOS | <a href="#">Ruby on Rails</a> |
|          |                       |                               |

## Related links

- [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 Nodejs](#)
  - [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](#)

# Accelerate real-time big data analytics with Spark connector for Azure SQL Database and SQL Server

9/25/2018 • 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

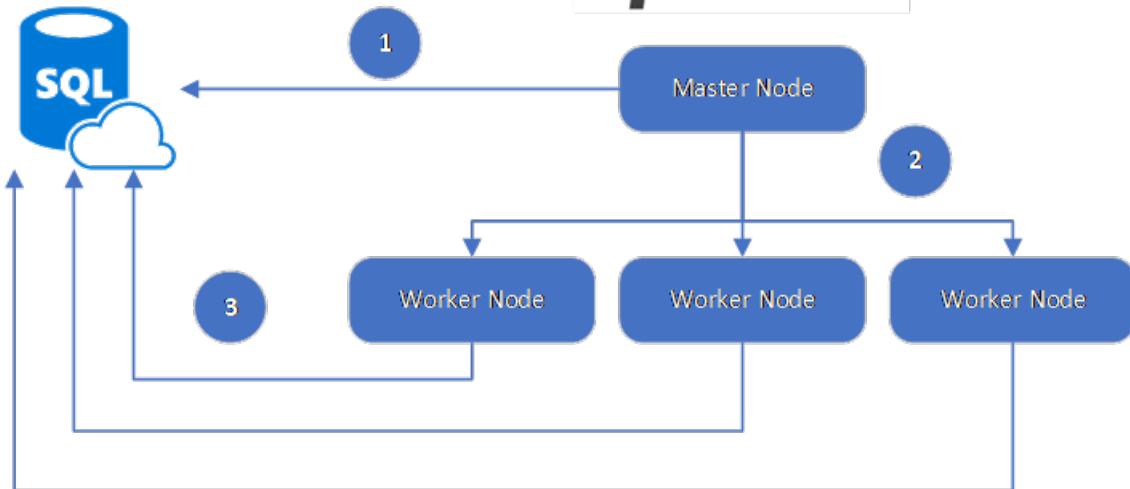
| COMPONENT                            | VERSION                  |
|--------------------------------------|--------------------------|
| Apache Spark                         | 2.0.2 or later           |
| Scala                                | 2.10 or later            |
| Microsoft JDBC Driver for SQL 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.



### 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"     -> "*****",
  "databaseName" -> "zeqisql",
  "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](#).

# Get the required values for authenticating an application to access SQL Database from code

10/23/2018 • 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

You need to have the latest [Azure PowerShell](#) installed and running. For detailed information, see [How to install and configure Azure PowerShell](#).

The following PowerShell script creates 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](#).

```
# Sign in to Azure.
Connect-AzureRmAccount

# If you have multiple subscriptions, uncomment and set to the subscription you want to work with.
#$subscriptionId = "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}"
#Set-AzureRmContext -SubscriptionId $subscriptionId

# Provide these values for your new AAD app.
# $appName is the display name for your app, must be unique in your directory.
# $uri does not need to be a real uri.
# $secret is a password you create.

$appName = "{app-name}"
$uri = "http://{app-name}"
$secret = "{app-password}"

# Create a AAD app
$azureAdApplication = New-AzureRmADApplication -DisplayName $appName -HomePage $Uri -IdentifierUris $Uri -
Password $secret

# Create a Service Principal for the app
$svcprincipal = New-AzureRmADServicePrincipal -ApplicationId $azureAdApplication.ApplicationId

# To avoid a PrincipalNotFound error, I pause here for 15 seconds.
Start-Sleep -s 15

# If you still get a PrincipalNotFound error, then rerun the following until successful.
$roleassignment = New-AzureRmRoleAssignment -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-AzureRmContext).Subscription.SubscriptionId
Write-Output "_tenantId:" (Get-AzureRmContext).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](#)

# SQL error codes for SQL Database client applications: Database connection errors and other issues

9/24/2018 • 19 minutes to read • [Edit Online](#)

This article lists SQL error codes for SQL Database client applications, including database connection errors, transient errors (also called transient faults), resource governance errors, database copy issues, elastic pool, and other errors. Most categories are particular to Azure SQL Database, and do not apply to Microsoft SQL Server. See also [system error messages](#).

## Database connection errors, transient errors, and other temporary errors

The following table covers the SQL error codes for connection loss errors, and other transient errors you might encounter when your application attempts to access SQL Database. For getting started tutorials on how to connect to Azure SQL Database, see [Connecting to Azure SQL Database](#).

### Most common database connection errors and transient fault errors

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*.

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.

Transient fault errors typically manifest as one of the following error messages from your client programs:

- Database <db\_name> on server <Azure\_instance> is not currently available. Please retry the connection later. If the problem persists, contact customer support, and provide them the session tracing ID of <session\_id>
- Database <db\_name> on server <Azure\_instance> is not currently available. Please retry the connection later. If the problem persists, contact customer support, and provide them the session tracing ID of <session\_id>. (Microsoft SQL Server, Error: 40613)
- An existing connection was forcibly closed by the remote host.
- System.Data.Entity.Core.EntityCommandExecutionException: An error occurred while executing the command definition. See the inner exception for details. ---> System.Data.SqlClient.SqlException: A transport-level error has occurred when receiving results from the server. (provider: Session Provider, error: 19 - Physical connection is not usable)
- An connection attempt to a secondary database failed because the database is in the process of reconfiguration and it is busy applying new pages while in the middle of an active transaction on the primary database.

For code examples of retry logic, see:

- [Connection Libraries for SQL Database and SQL Server](#)
- [Actions to fix connection errors and transient errors in SQL Database](#)

A discussion of the *blocking period* for clients that use ADO.NET is available in [SQL Server Connection Pooling \(ADO.NET\)](#).

## Transient fault error codes

The following errors are transient, and should be retried in application logic:

| ERROR CODE | SEVERITY | DESCRIPTION  |
|------------|----------|--|
| 4060       | 16       | Cannot open database "%.*ls" requested by the login. The login failed.   |
| 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.</p> |
| 40501      | 20       | <p>The service is currently busy. Retry the request after 10 seconds. Incident ID: %ls. Code: %d.</p> <p>For more information, see:</p> <ul style="list-style-type: none"> <li>• <a href="#">Azure SQL Database resource limits</a>.</li> </ul>  |
| 40613      | 17       | 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'.   |
| 49918      | 16       | <p>Cannot process request. Not enough resources to process request.</p> <p>The service is currently busy. Please retry the request later.</p>  |

| ERROR CODE | SEVERITY | DESCRIPTION   |
|------------|----------|---|
| 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.</p> |
| 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.</p>   |
| 4221       | 16       | <p>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.</p>                               |

## 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.             |

| ERROR CODE | SEVERITY | DESCRIPTION   |
|------------|----------|---|
| 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.   |

## Resource governance errors

The following errors are caused by excessive use of resources while working with Azure SQL Database. For example:

- A transaction has been open for too long.
- A transaction is holding too many locks.
- An application is consuming too much memory.
- An application is consuming too much `TempDb` space.

Related topics:

- More detailed information is available here: [Azure SQL Database resource limits](#).

| 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="http://go.microsoft.com/fwlink/?LinkId=267637">http://go.microsoft.com/fwlink/?LinkId=267637</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="#">Azure SQL Database resource limits</a>.</li> </ul>  |
| 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. For more information, see <a href="http://go.microsoft.com/fwlink/?LinkId=267637">http://go.microsoft.com/fwlink/?LinkId=267637</a>. Otherwise, please try again later.</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="#">Azure SQL Database resource limits</a>.</li> </ul> |
| 40544      | 20       | <p>The database has reached its size quota. Partition or delete data, drop indexes, or consult the documentation for possible resolutions.</p>   |
| 40549      | 16       | <p>Session is terminated because you have a long-running transaction. Try shortening your transaction.</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.</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.</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.</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.</p> |
| 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.</p>   |

## Elastic pool errors

The following errors are related to creating and using elastic pools:

| ERRONNUMBER | ERRORSEVERITY | ERRORFORMAT  | ERRORINSERTS                     | ERRORCAUSE  | ERRORCORRECTIVEACTION   |
|-------------|---------------|--|----------------------------------|---|---|
| 1132        | EX_RESOURCE   | The elastic pool has reached its storage limit. The storage usage for the elastic pool cannot exceed (%d) MBs. | Elastic pool space limit in MBs. | Attempting to write data to a database when the storage limit of the elastic pool has been reached. | 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. |

| ERRORNUMBER | ERRORSEVERITY | ERRORFORMAT  | ERRORINSERTS   | ERRORCAUSE   | ERRORCORRECTIVEACTION  |
|-------------|---------------|--|--|--|--|
| 10929       | EX_USER       | 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 <a href="http://go.microsoft.com/fwlink/?LinkId=267637">http://go.microsoft.com/fwlink/?LinkId=267637</a> for assistance. 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. | 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. |
| 40844       | EX_USER       | Database '%ls' on Server '%ls' is a '%ls' edition database in an elastic pool and cannot have a continuous copy relationship.  | database name, database edition, server name               | A StartDatabaseCopy command is issued for a non-premium db in an elastic pool.   | Coming soon  |
| 40857       | EX_USER       | Elastic pool not found for server: '%ls', elastic pool name: '%ls'.  | name of server; elastic pool name                          | Specified elastic pool does not exist in the specified server.   | Provide a valid elastic pool name.   |
| 40858       | EX_USER       | Elastic pool '%ls' already exists in server: '%ls'   | elastic pool name, server name                             | Specified elastic pool already exists in the specified logical server.   | Provide new elastic pool name.   |
| 40859       | EX_USER       | Elastic pool does not support service tier '%ls'.  | elastic pool service tier                                  | Specified service tier is not supported for elastic pool provisioning.   | Provide the correct edition or leave service tier blank to use the default service tier.   |
| 40860       | EX_USER       | Elastic pool '%ls' and service objective '%ls' combination is invalid.   | elastic pool name; service tier                            | Elastic pool and service tier can be specified together only if resource type is specified as 'ElasticPool'.                   | Specify correct combination of elastic pool and service tier.  |

| ERRORNUMBER | ERRORSEVERITY | ERRORFORMAT   | ERRORINSERTS  | ERRORCAUSE   | ERRORCORRECTIVEACTION  |
|-------------|---------------|---|---|--|--|
| 40861       | EX_USER       | The database edition '%.ls' cannot be different than the elastic pool service tier which is '%.ls'.         | database edition, elastic pool service tier                                   | 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       | EX_USER       | Elastic pool name must be specified if the elastic pool service objective is specified.                     | None  | Elastic pool service objective does not uniquely identify an elastic pool.                                 | Specify the elastic pool name if using the elastic pool service objective.   |
| 40864       | EX_USER       | The DTUs for the elastic pool must be at least (%d) DTUs for service tier '%.*ls'.                          | DTUs for elastic pool; elastic pool service tier.                             | 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       | EX_USER       | The DTUs for the elastic pool cannot exceed (%d) DTUs for service tier '%.*ls'.                             | DTUs for elastic pool; elastic pool service tier.                             | 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       | EX_USER       | The DTU max per database must be at least (%d) for service tier '%.*ls'.                                    | DTU max per database; elastic pool service tier                               | 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       | EX_USER       | The DTU max per database cannot exceed (%d) for service tier '%.*ls'.                                       | DTU max per database; elastic pool service tier.                              | 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       | EX_USER       | The DTU min per database cannot exceed (%d) for service tier '%.*ls'.                                       | DTU min per database; elastic pool service tier.                              | 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       | EX_USER       | The number of databases (%d) and DTU min per database (%d) cannot exceed the DTUs of the elastic pool (%d). | Number databases in elastic pool; DTU min per database; DTUs of elastic pool. | 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.       |

| ERRORNUMBER | ERRORSEVERITY | ERRORFORMAT   | ERRORINSERTS   | ERRORCAUSE   | ERRORCORRECTIVEACTION   |
|-------------|---------------|---|--|--|---|
| 40877       | EX_USER       | An elastic pool cannot be deleted unless it does not contain any databases.   | None   | 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       | EX_USER       | 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. | Name of elastic pool; database count limit of elastic pool; eDTUs for resource pool. | 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       | EX_USER       | The DTUs or storage limit for the elastic pool '%.*ls' cannot be decreased since that would not provide sufficient storage space for its databases.                 | Name of elastic pool.  | 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       | EX_USER       | The DTU min per database (%d) cannot exceed the DTU max per database (%d).  | DTU min per database; DTU max per database.  | 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         | EX_USER       | The storage size for an individual database in an elastic pool cannot exceed the max size allowed by '%.*ls' service tier elastic pool.                             | elastic pool service tier  | 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.  |

Related topics:

- [Create an elastic pool \(C#\)](#)
- [Manage an elastic pool \(C#\)](#).
- [Create an elastic pool \(PowerShell\)](#)
- [Monitor and manage an elastic pool \(PowerShell\)](#).

## General errors

The following errors do not fall into any previous categories.

| ERROR CODE | SEVERITY | DESCRIPTION   |
|------------|----------|---|
| 15006      | 16       | (AdministratorLogin) is not a valid name because it contains invalid characters.  |
| 18452      | 14       | Login failed. The login is from an untrusted domain and cannot be used with Windows authentication.%.*ls (Windows logins are not supported in this version of SQL Server.)              |
| 18456      | 14       | Login failed for user '%.*ls'.%.*ls%.%.*ls(The login failed for user "%.*ls". The password change failed. Password change during login is not supported in this version of SQL Server.) |
| 18470      | 14       | Login failed for user '%.*ls'. Reason: The account is disabled.%.*ls  |
| 40014      | 16       | Multiple databases cannot be used in the same transaction.  |
| 40054      | 16       | Tables without a clustered index are not supported in this version of SQL Server. Create a clustered index and try again.   |
| 40133      | 15       | This operation is not supported in this version of SQL Server.  |
| 40506      | 16       | Specified SID is invalid for this version of SQL Server.  |
| 40507      | 16       | '%.*ls' cannot be invoked with parameters in this version of SQL Server.  |
| 40508      | 16       | USE statement is not supported to switch between databases. Use a new connection to connect to a different database.  |
| 40510      | 16       | Statement '%.*ls' is not supported in this version of SQL Server  |
| 40511      | 16       | Built-in function '%.*ls' is not supported in this version of SQL Server.   |
| 40512      | 16       | Deprecated feature '%ls' is not supported in this version of SQL Server.  |

| ERROR CODE | SEVERITY | DESCRIPTION  |
|------------|----------|--|
| 40513      | 16       | Server variable '%.*ls' is not supported in this version of SQL Server.  |
| 40514      | 16       | '%ls' is not supported in this version of SQL Server.  |
| 40515      | 16       | Reference to database and/or server name in '%.*ls' is not supported in this version of SQL Server.                        |
| 40516      | 16       | Global temp objects are not supported in this version of SQL Server.   |
| 40517      | 16       | Keyword or statement option '%.*ls' is not supported in this version of SQL Server.  |
| 40518      | 16       | DBCC command '%.*ls' is not supported in this version of SQL Server.   |
| 40520      | 16       | Securable class '%S_MSG' not supported in this version of SQL Server.  |
| 40521      | 16       | Securable class '%S_MSG' not supported in the server scope in this version of SQL Server.                                  |
| 40522      | 16       | Database principal '%.*ls' type is not supported in this version of SQL Server.  |
| 40523      | 16       | Implicit user '%.*ls' creation is not supported in this version of SQL Server. Explicitly create the user before using it. |
| 40524      | 16       | Data type '%.*ls' is not supported in this version of SQL Server.  |
| 40525      | 16       | WITH '%.ls' is not supported in this version of SQL Server.  |
| 40526      | 16       | '%.*ls' rowset provider not supported in this version of SQL Server.   |
| 40527      | 16       | Linked servers are not supported in this version of SQL Server.  |
| 40528      | 16       | Users cannot be mapped to certificates, asymmetric keys, or Windows logins in this version of SQL Server.                  |
| 40529      | 16       | Built-in function '%.*ls' in impersonation context is not supported in this version of SQL Server.                         |

| ERROR CODE | SEVERITY | DESCRIPTION  |
|------------|----------|--|
| 40532      | 11       | Cannot open server "%.*ls" requested by the login. The login failed.   |
| 40553      | 16       | The session has been terminated because of excessive memory usage. Try modifying your query to process fewer rows.<br><br>Reducing the number of <code>ORDER BY</code> and <code>GROUP BY</code> operations in your Transact-SQL code helps reduce the memory requirements of your query.  |
| 40604      | 16       | Could not CREATE/ALTER DATABASE because it would exceed the quota of the server.   |
| 40606      | 16       | Attaching databases is not supported in this version of SQL Server.  |
| 40607      | 16       | Windows logins are not supported in this version of SQL Server.  |
| 40611      | 16       | Servers can have at most 128 firewall rules defined.   |
| 40614      | 16       | Start IP address of firewall rule cannot exceed End IP address.  |
| 40615      | 16       | Cannot open server '{0}' requested by the login. Client with IP address '{1}' is not allowed to access the server.<br><br>To enable access, use the SQL Database Portal or run <code>sp_set_firewall_rule</code> on the master database to create a firewall rule for this IP address or address range. It may take up to five minutes for this change to take effect. |
| 40617      | 16       | The firewall rule name that starts with (rule name) is too long. Maximum length is 128.  |
| 40618      | 16       | The firewall rule name cannot be empty.  |
| 40620      | 16       | The login failed for user "%.*ls". The password change failed. Password change during login is not supported in this version of SQL Server.  |
| 40627      | 20       | Operation on server '{0}' and database '{1}' is in progress. Please wait a few minutes before trying again.  |

| ERROR CODE | SEVERITY | DESCRIPTION   |
|------------|----------|---|
| 40630      | 16       | Password validation failed. The password does not meet policy requirements because it is too short.   |
| 40631      | 16       | The password that you specified is too long. The password should have no more than 128 characters.  |
| 40632      | 16       | Password validation failed. The password does not meet policy requirements because it is not complex enough.  |
| 40636      | 16       | Cannot use a reserved database name '%.*ls' in this operation.  |
| 40638      | 16       | Invalid subscription id (subscription-id). Subscription does not exist.   |
| 40639      | 16       | Request does not conform to schema: (schema error).   |
| 40640      | 20       | The server encountered an unexpected exception.   |
| 40641      | 16       | The specified location is invalid.  |
| 40642      | 17       | The server is currently too busy. Please try again later.   |
| 40643      | 16       | The specified x-ms-version header value is invalid.   |
| 40644      | 14       | Failed to authorize access to the specified subscription.   |
| 40645      | 16       | Servername (servername) cannot be empty or null. It can only be made up of lowercase letters 'a'-'z', the numbers 0-9 and the hyphen. The hyphen may not lead or trail in the name. |
| 40646      | 16       | Subscription ID cannot be empty.  |
| 40647      | 16       | Subscription (subscription-id) does not have server (servername).   |
| 40648      | 17       | Too many requests have been performed. Please retry later.  |
| 40649      | 16       | Invalid content-type is specified. Only application/xml is supported.   |

| ERROR CODE | SEVERITY | DESCRIPTION  |
|------------|----------|--|
| 40650      | 16       | Subscription (subscription-id) does not exist or is not ready for the operation.   |
| 40651      | 16       | Failed to create server because the subscription (subscription-id) is disabled.  |
| 40652      | 16       | Cannot move or create server. Subscription (subscription-id) will exceed server quota.   |
| 40671      | 17       | Communication failure between the gateway and the management service. Please retry later.  |
| 40852      | 16       | Cannot open database '%.*ls' on server '%.*ls' requested by the login. Access to the database is only allowed using a security-enabled connection string. To access this database, modify your connection strings to contain 'secure' in the server FQDN - 'server name'.database.windows.net should be modified to 'server name'.database.<br>secure.windows.net. |
| 40914      | 16       | Cannot open server '[server-name]' requested by the login. Client is not allowed to access the server.<br><br>To fix, consider adding a <a href="#">virtual network rule</a> .   |
| 45168      | 16       | The SQL Azure system is under load, and is placing an upper limit on concurrent DB CRUD operations for a single server (e.g., create database). The server specified in the error message has exceeded the maximum number of concurrent connections. Try again later.  |
| 45169      | 16       | The SQL azure system is under load, and is placing an upper limit on the number of concurrent server CRUD operations for a single subscription (e.g., create server). The subscription specified in the error message has exceeded the maximum number of concurrent connections, and the request was denied. Try again later.                                      |

## Next steps

- Read about [Azure SQL Database features](#).
- Read about [DTU-based purchasing model](#).
- Read about [vCore-based purchasing model](#).

# How to use batching to improve SQL Database application performance

9/25/2018 • 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 NavHistoryEventArgs : System.EventArgs
{
    public NavHistoryEventArgs(NavHistoryData data) { Data = data; }
    public NavHistoryData Data { get; set; }
}

public class NavHistoryDataMonitor
{
    public event EventHandler<NavHistoryEventArgs> OnAdded;

    public NavHistoryDataMonitor()
    {
        var observableData =
            Observable.FromEventPattern<NavHistoryEventArgs>(this, "OnAdded");

        observableData.Buffer(TimeSpan.FromSeconds(20), 1000).Subscribe(Handler);
    }

    public void RecordUserNavigationEntry(NavHistoryData data)
    {
        if (OnAdded != null)
            OnAdded(this, new NavHistoryEventArgs(data));
    }

    protected void Handler(IList<EventPattern<NavHistoryEventArgs>> 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<NavHistoryEventArgs> 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](#).

# Troubleshoot, diagnose, and prevent SQL connection errors and transient errors for SQL Database

9/24/2018 • 15 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**. Compare the **Number** property against the list of transient errors that are found near the top of the article [SQL error codes for SQL Database client applications](#).

### 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, your program can correct the misspelling 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\_MyUserName'."

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.
- Purposely 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 1 second. 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.Execute` 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.

- On the blade for your database, select **Show database connection strings**.
- 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 left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Configure, Geo-Replication, and Connection strings (which is highlighted). The main content area displays three connection string formats under the ADO.NET tab:

- 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.

- 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.
- 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.

- Sign in to the [Azure portal](#).
- In the list on the left, select **All services**.
- Scroll and select **SQL servers**.

The screenshot shows the 'All services' blade in the Microsoft Azure portal. On the left, there's a sidebar with various service icons. The 'SQL servers' service is selected and highlighted with a blue border. Other services listed in the main pane include:

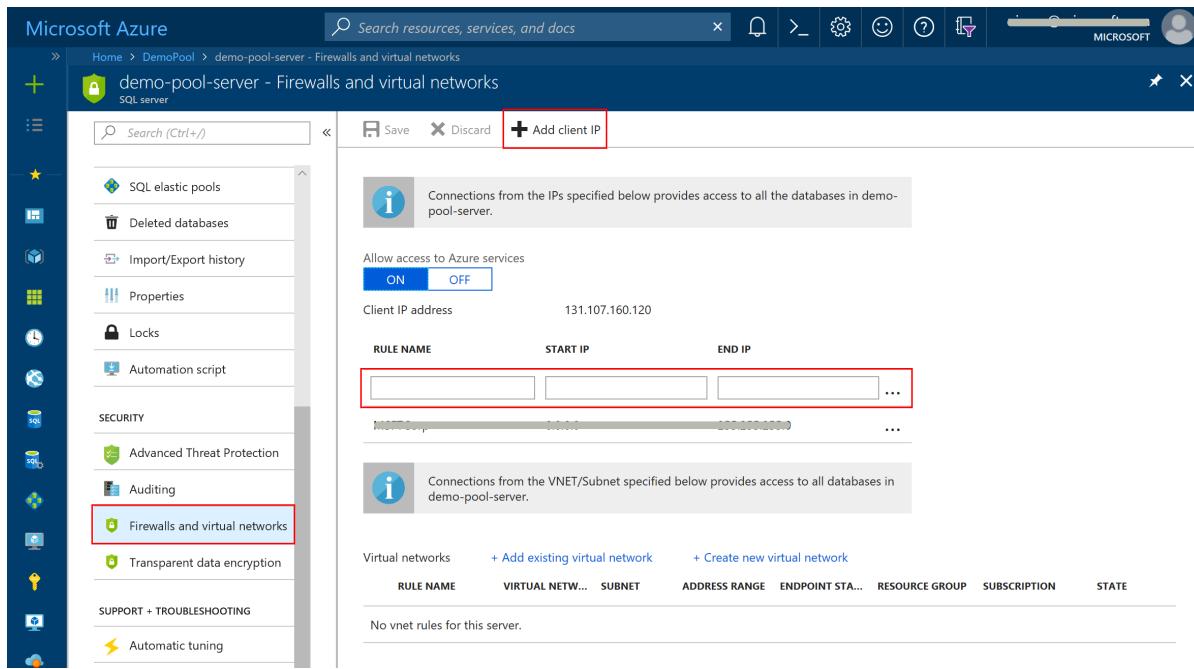
- Azure Cosmos DB
- SQL databases
- Azure Database for PostgreSQL servers
- SQL data warehouses
- Redis Caches
- Data factories
- Virtual clusters
- Elastic Job agents

A 'Filter' text box is at the top left, and there are 'Collapse all' and 'Expand all' buttons at the top right.

- In the filter text box, start typing the name of your server. Your row is displayed.
- 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 for managing a SQL server named 'demo-pool-server'. In the left sidebar, under the 'SECURITY' section, the 'Firewalls and virtual networks' item is selected and highlighted with a red box. At the top right, there is a red box around the '+ Add client IP' button. Below it, there is an information icon with a message about connections from specified IPs. A switch labeled 'Allow access to Azure services' is set to 'ON'. Under 'Client IP address', the value '131.107.160.120' is listed. A table titled 'RULE NAME' shows a single row with three columns: 'RULE NAME' (containing a red box), 'START IP', and 'END IP'. The 'START IP' and 'END IP' fields show ranges: '131.107.160.120' and '131.107.160.255'. Below this table, another information icon mentions connections from VNET/Subnet. A table for 'Virtual networks' is shown with columns: 'RULE NAME', 'VIRTUAL NETW...', 'SUBNET', 'ADDRESS RANGE', 'ENDPOINT STA...', 'RESOURCE GROUP', 'SUBSCRIPTION', and 'STATE'. A note at the bottom states 'No vnet rules for this server.'

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.*<br/>FROM sys.event_log AS e<br/>WHERE e.database_name = 'myDbName'<br/>AND e.event_category = 'connectivity'<br/>AND 2 &gt;= DateDiff<br/>    (hour, e.end_time, GetUtcDate())<br/>ORDER BY e.event_category,<br/>        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.*<br/>FROM sys.database_connection_stats AS c<br/>WHERE c.database_name = 'myDbName'<br/>AND 24 &gt;= DateDiff<br/>    (hour, c.end_time, GetUtcDate())<br/>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

- For more information on troubleshooting other common SQL Database connection issues, see [Troubleshoot connection issues to Azure SQL Database](#).
- [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

10/19/2018 • 4 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.

## 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 than 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

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-AzureRMSqlServerDNSAlias](#): Creates a new DNS alias in the Azure SQL Database service system. The alias refers to Azure SQL Database server 1.
- [Get-AzureRMSqlServerDNSAlias](#): Get and list all the DNS aliases that are assigned to SQL DB server 1.
- [Set-AzureRMSqlServerDNSAlias](#): Modifies the server name that the alias is configured to refer to, from server 1 to SQL DB server 2.
- [Remove-AzureRMSqlServerDNSAlias](#): Remove the DNS alias from SQL DB server 2, by using the name of the alias.

The preceding cmdlets were added to the **AzureRM.Sql** module starting with module version 5.1.1.

## 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

10/19/2018 • 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. The script runs the following cmdlets which takes the following actions:

The cmdlets used in the code example are the following:

- [New-AzureRMSqlServerDNSAlias](#): Creates a new DNS alias in the Azure SQL Database service system. The alias refers to Azure SQL Database server 1.
- [Get-AzureRMSqlServerDNSAlias](#): Get and list all the DNS aliases that are assigned to SQL DB server 1.
- [Set-AzureRMSqlServerDNSAlias](#): Modifies the server name that the alias is configured to refer to, from server 1 to SQL DB server 2.
- [Remove-AzureRMSqlServerDNSAlias](#): Remove the DNS alias from SQL DB server 2, by using the name of the alias.

The preceding PowerShell cmdlets were added to the **AzureRm.Sql** module starting with version 5.1.1.

## 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:

- Example server string: `any-unique-alias-name.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 a free trial, click <https://azure.microsoft.com/free/>.
- Azure PowerShell module, with cmdlet **New-AzureRMSqlServerDNSAlias**.
  - To install or upgrade, see [Install Azure PowerShell module](#).
  - Run `Get-Module -ListAvailable AzureRM;` in powershell\_ise.exe, to find the version.
- Two Azure SQL Database servers.

## Code example

The following PowerShell code example starts by assign literal values to several variables. To run the code, you must first edit all the placeholder values to match real values in your system. Or you can just study the code. And the console output of the code is also provided.

```

#####
### Assign prerequisites. #####
#####

cls;

$SubscriptionName          = '<EDIT-your-subscription-name>';
[string]$SubscriptionGuid_Get = '?'; # The script assigns this value, not you.

$SqlServerDnsAliasName = '<EDIT-any-unique-alias-name>';

$1ResourceGroupName = '<EDIT-rg-1>'; # Can be same or different than $2ResourceGroupName.
$1SqlServerName      = '<EDIT-sql-1>'; # Must differ from $2SqlServerName.

$2ResourceGroupName = '<EDIT-rg-2>';
$2SqlServerName     = '<EDIT-sql-2>';

# Login to your Azure subscription, first time per session.
Write-Host "You must log into Azure once per powershell_ise.exe session,";
Write-Host " thus type 'yes' only the first time.";
Write-Host " ";
$yesno = Read-Host '[yes/no] Do you need to log into Azure now?';
if ('yes' -eq $yesno)
{
    Connect-AzureRmAccount -SubscriptionName $SubscriptionName;
}

$SubscriptionGuid_Get = Get-AzureRmSubscription `

-SubscriptionName $SubscriptionName;

#####
### Working with DNS aliasing for Azure SQL DB server. #####
#####

Write-Host '[1] Assign a DNS alias to SQL DB server 1.';
New-AzureRMSqlServerDNSAlias `

-ResourceGroupName $1ResourceGroupName `

-ServerName       $1SqlServerName `

-ServerDNSAliasName $SqlServerDnsAliasName;

Write-Host '[2] Get and list all the DNS aliases that are assigned to SQL DB server 1.';
Get-AzureRMSqlServerDNSAlias `

-ResourceGroupName $1ResourceGroupName `

-ServerName       $1SqlServerName;

Write-Host '[3] Move the DNS alias from 1 to SQL DB server 2.';

Set-AzureRMSqlServerDNSAlias `

-ResourceGroupName $2ResourceGroupName `

-NewServerName    $2SqlServerName `

-ServerDNSAliasName $SqlServerDnsAliasName `

-OldServerResourceGroup $1ResourceGroupName `

-OldServerName     $1SqlServerName `

-OldServerSubscriptionId $SubscriptionGuid_Get;

# Here your client, such as SSMS, can connect to your "$2SqlServerName"
# by using "$SqlServerDnsAliasName" in the server name.
# For example, server: "any-unique-alias-name.database.windows.net".

# Remove-AzureRMSqlServerDNSAlias - would fail here for SQL DB server 1.

Write-Host '[4] Remove the DNS alias from SQL DB server 2.';

Remove-AzureRMSqlServerDNSAlias `

-ResourceGroupName $2ResourceGroupName `

-ServerName       $2SqlServerName `

-ServerDNSAliasName $SqlServerDnsAliasName;

```

## Actual console output from the PowerShell example

The following console output was copied and pasted from an actual run.

```
You must log into Azure once per powershell_ise.exe session,
thus type 'yes' only the first time.

[yes/no] Do you need to log into Azure now?: yes

Environment      : AzureCloud
Account         : gm@acorporation.com
TenantId        : 72f988bf-1111-1111-1111-111111111111
SubscriptionId  : 45651c69-2222-2222-2222-222222222222
SubscriptionName : mysubscriptionname
CurrentStorageAccount :

[1] Assign a DNS alias to SQL DB server 1.
[2] Get the DNS alias that is assigned to SQL DB server 1.
[3] Move the DNS alias from 1 to SQL DB server 2.
[4] Remove the DNS alias from SQL DB server 2.

ResourceGroupName ServerName      ServerDNSAliasName
-----          -----
gm-rg-dns-1     gm-sqldb-dns-1    unique-alias-name-food
gm-rg-dns-1     gm-sqldb-dns-1    unique-alias-name-food
gm-rg-dns-2     gm-sqldb-dns-2    unique-alias-name-food

[C:\windows\system32\]
>>
```

## 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

10/16/2018 • 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 or 14000-14999.
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 and 14000-14999 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://blogs.msdn.microsoft.com/sqlcat/2016/09/08/connect-to-azure-sql-database-v12-via-redirection/>
- **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++

9/24/2018 • 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        | Show database connection strings  |
| Subscription name | Pricing tier  |
|                   | S2 Standard (50 DTUs)   |

### Database connection strings

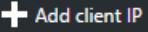
ADO.NET(SQL authentication)  
`MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;` 

ODBC (Includes Node.js) [SQL authentication]  
`Driver=(ODBC Driver 13 for SQL Server);Server=tcp:cppsqlDb.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 cppsqlDb.

Allow access to Azure services  ON  OFF

Client IP address 167.2~~0.0.0~~

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

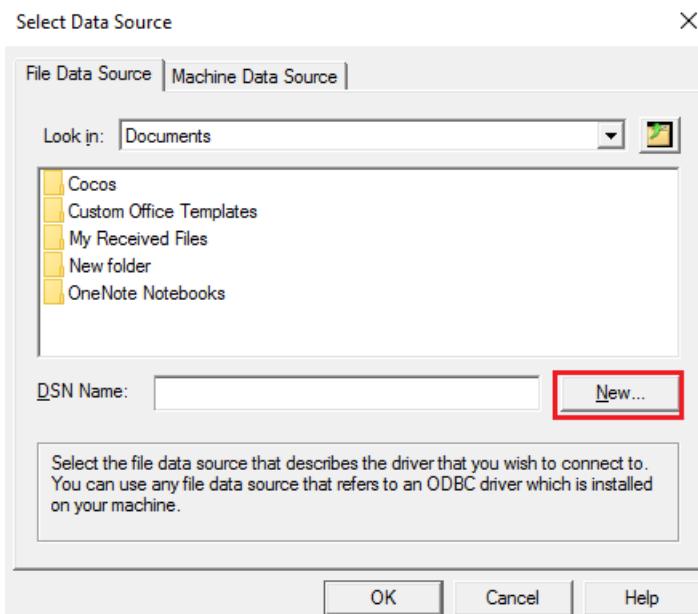
|                         |  |
|-------------------------|--|
| Command                 | \$(TargetPath)                                     |
| Command Arguments       | "Driver={SQL Server Native Client 11.0};Server=tcp |
| Working Directory       | \$(ProjectDir)                                     |
| Attach                  | No   |
| Debugger Type           | Auto   |
| Environment             |  |
| Merge Environment       | Yes  |
| SQL Debugging           | No   |
| Amp Default Accelerator | WARP software accelerator                          |

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

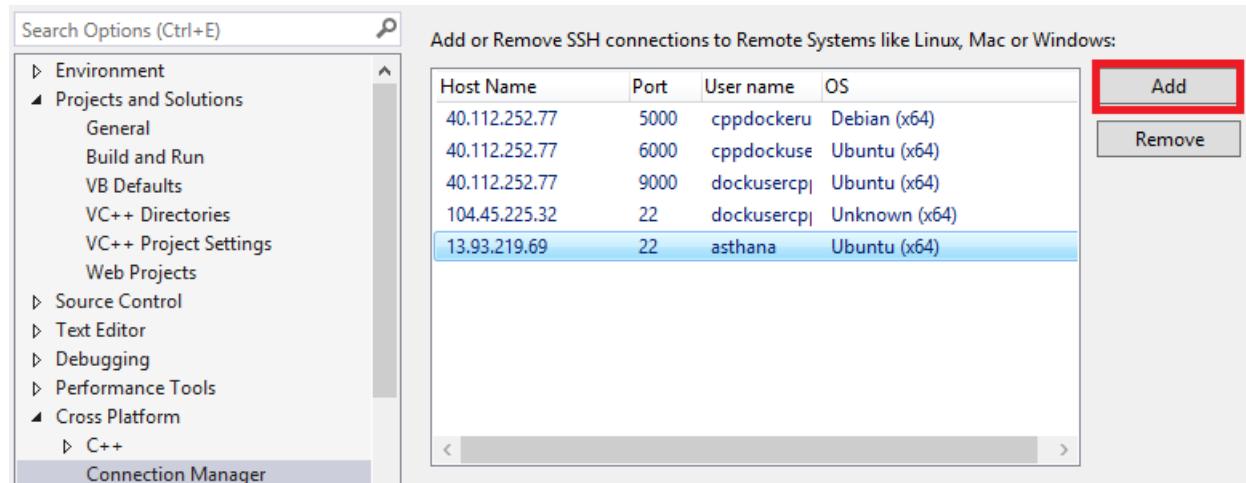
In case you haven't heard the news yet, Visual Studio now allows you to develop C++ Linux application as well. You can read about this new scenario in the [Visual C++ for Linux Development](#) blog. To build for Linux, you need a remote machine where your Linux distro is running. If you don't have one 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.

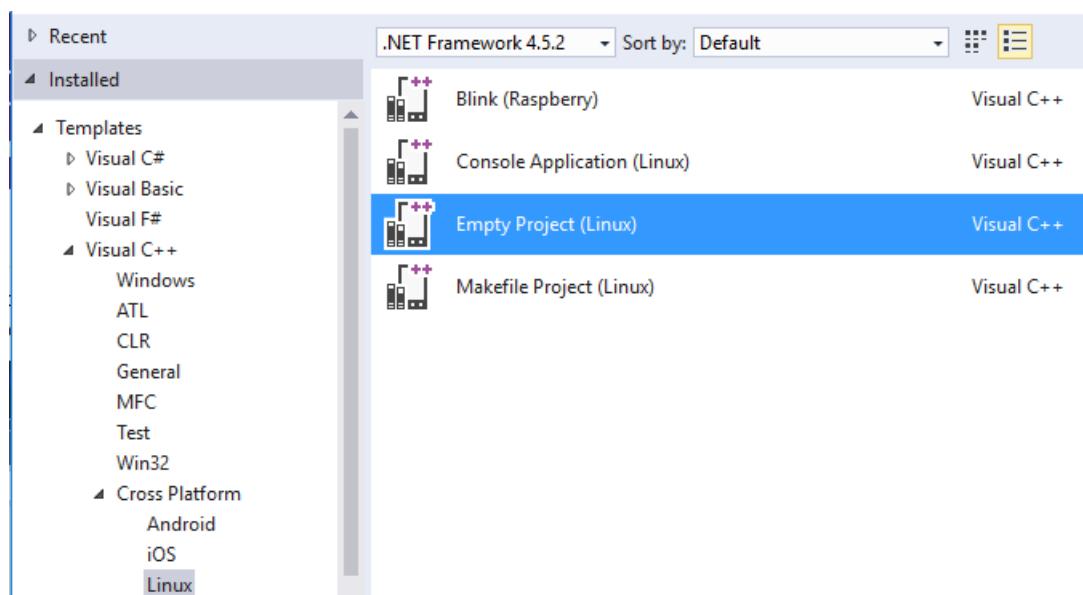
The following steps install the libraries needed for SQL and ODBC for your distro:

```
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:

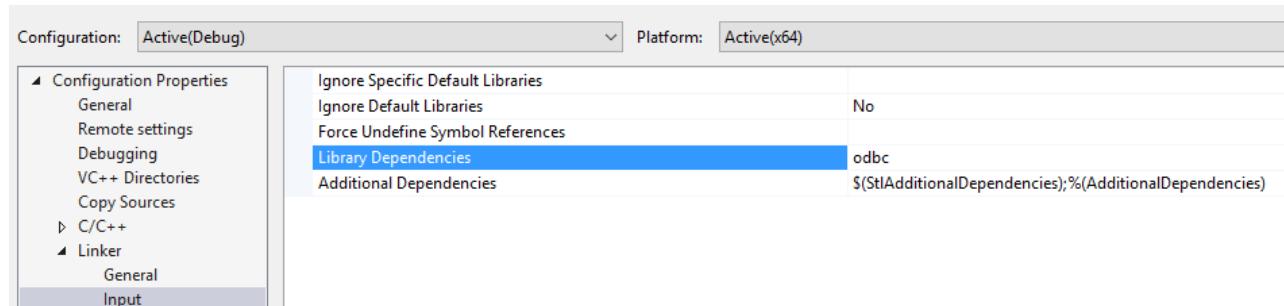


You can then add a [new C source file](#) and replace it with this content. Using the ODBC APIs `SQLAllocHandle`, `SQLSetConnectAttr`, and `SQLDriverConnect`, you should be able to initialize and establish a connection to your database. Like with the Windows ODBC sample, you need to replace the `SQLDriverConnect` call with the details

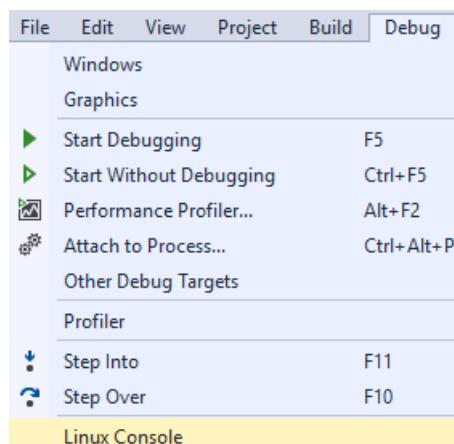
from your database connection string parameters copied from the Azure portal previously.

```
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);
```

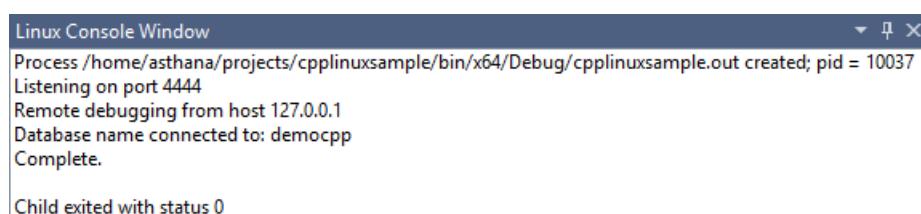
The last thing to do before compiling is to add **odbc** as a library dependency:



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:



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 an Azure SQL database and create a report

9/24/2018 • 4 minutes to read • [Edit Online](#)

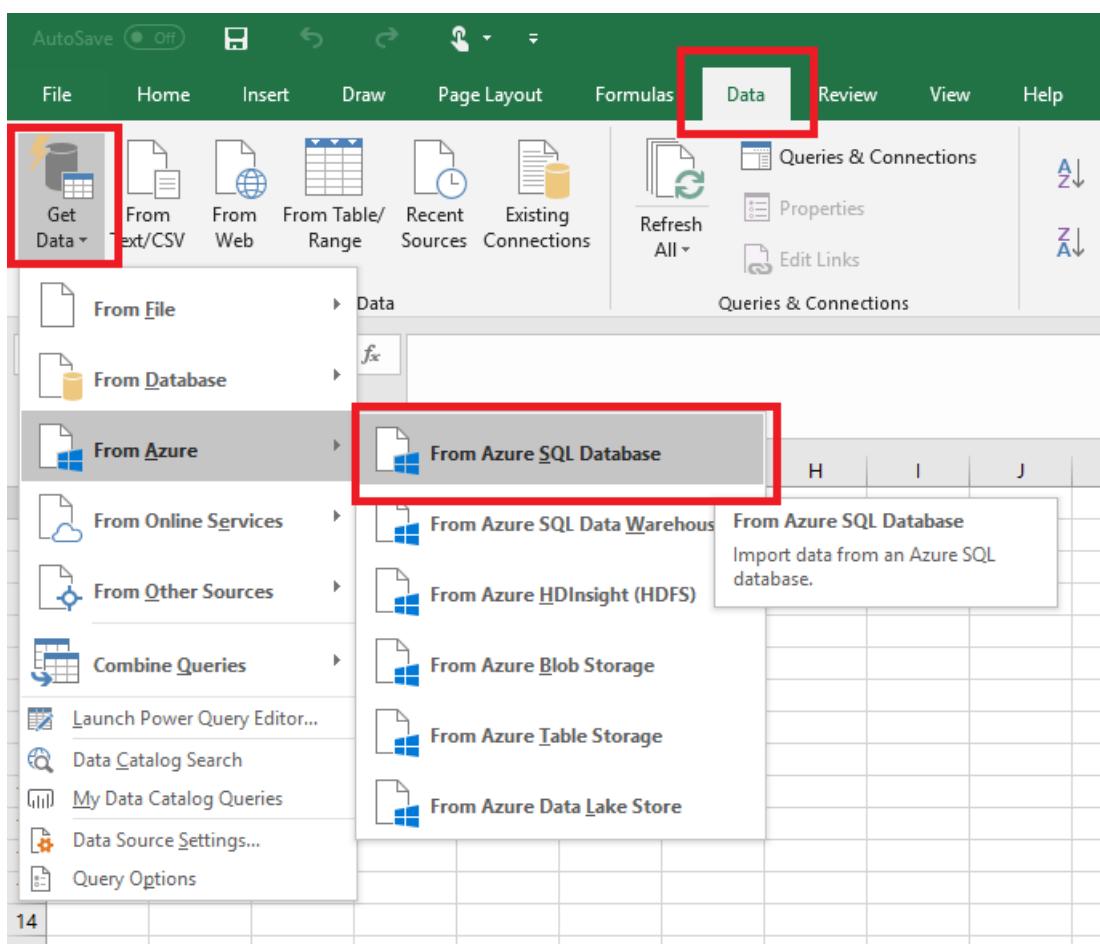
Connect Excel to a SQL database in the cloud and import data and create tables and charts based on values in the database. In this tutorial you will set up the connection between Excel and a database table, save the file that stores data and the connection information for Excel, and then create a pivot chart from the database values.

You'll need a SQL database in Azure before you get started. If you don't have one, see [Create your first SQL database](#) to get a 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

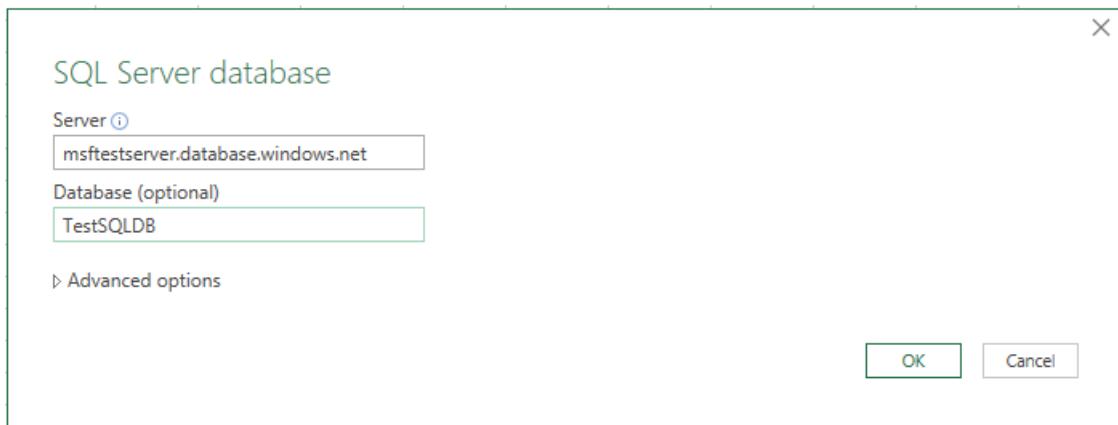
1. To connect Excel to SQL database, open Excel and then create a new workbook or open an existing Excel workbook.
2. In the menu bar at the top of the page, select the **Data** tab, select **Get Data**, select From Azure, and then select **From Azure SQL Database**.



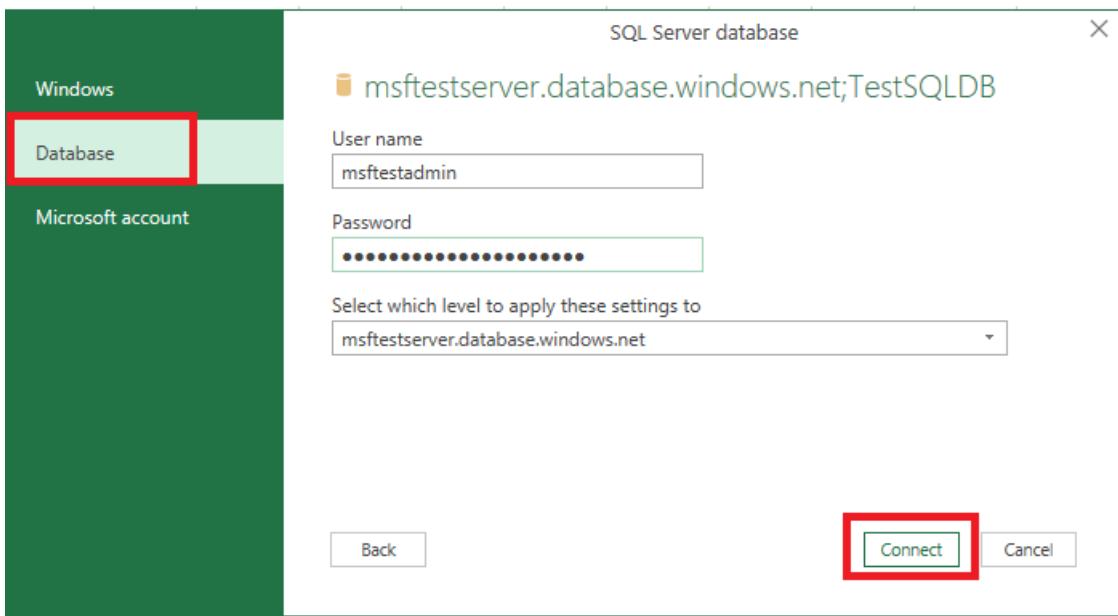
The Data Connection Wizard opens.

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.

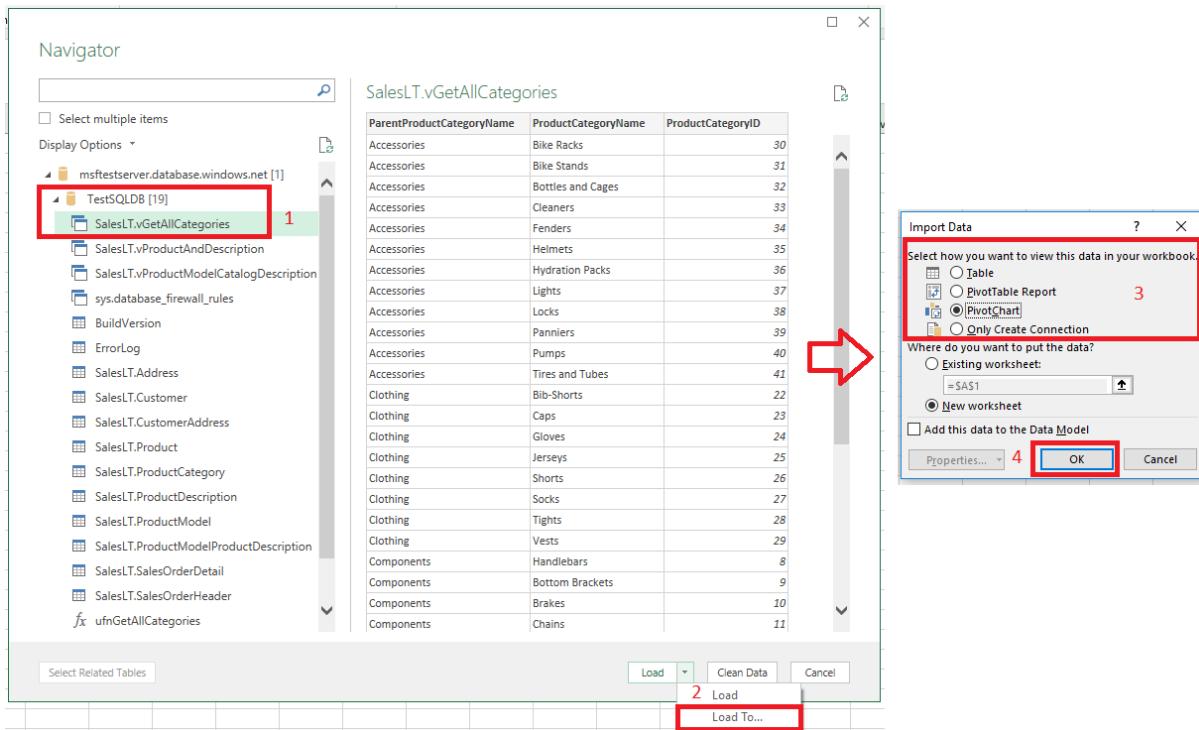
5. In the **Navigator**, select the database you want to work with from the list, select the tables or views you want to work with (we chose **vGetAllCategories**), and then select **Load** to move the data from your SQL Azure database to your excel spreadsheet.

| 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

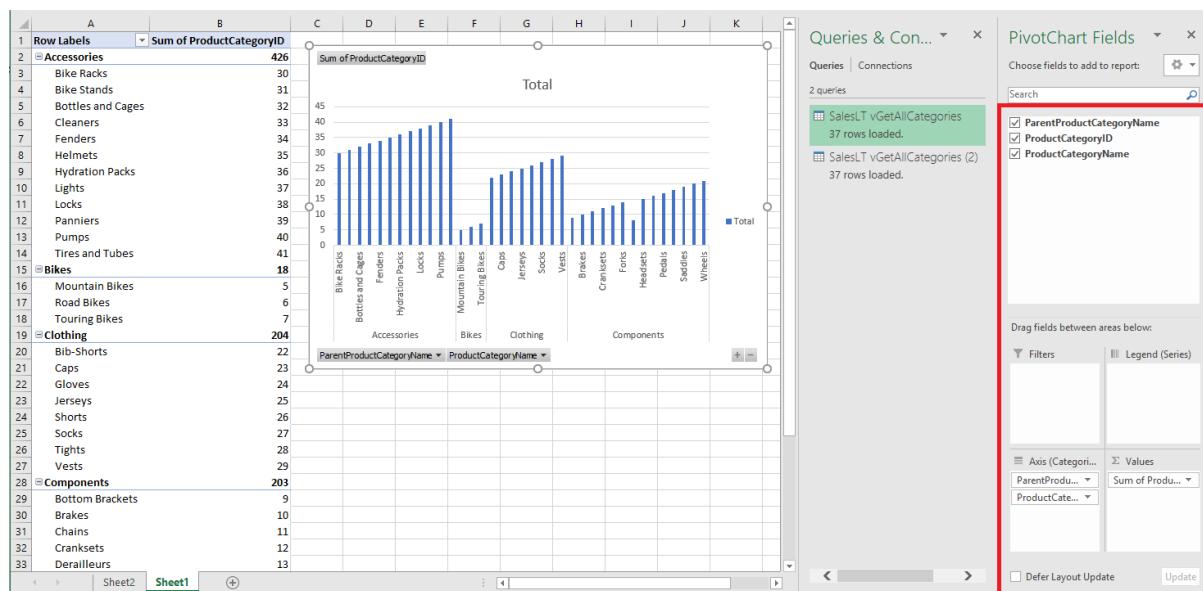
Now that you've established the connection, you have several different options with how to load the data. For example, the following steps create a pivot chart based on the data found in your SQL Database.

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](#).



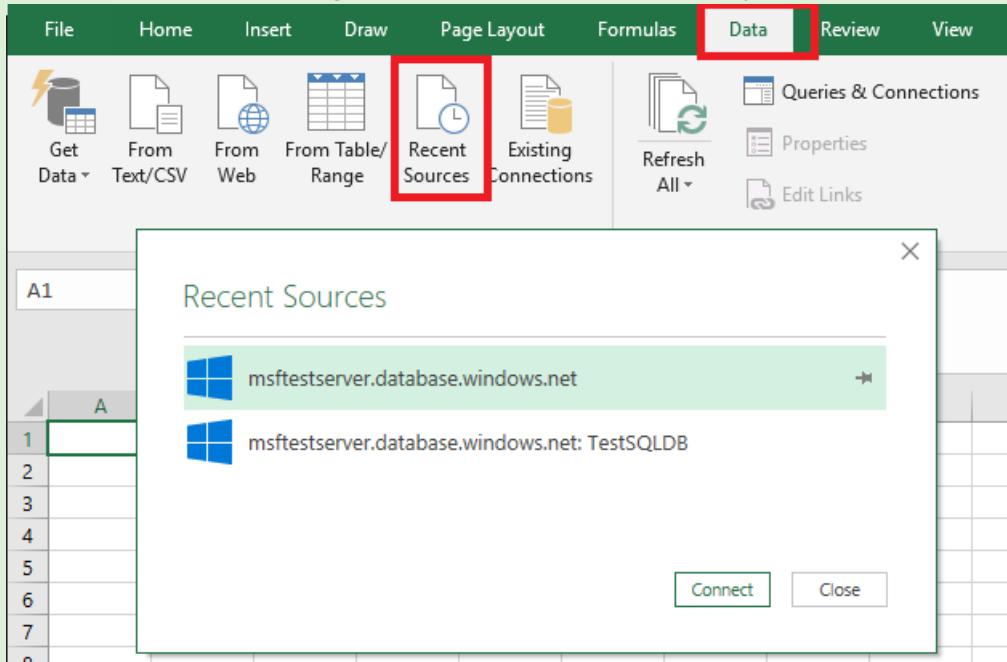
The worksheet now has an empty pivot table and chart.

### 3. Under **PivotTable Fields**, select all the check-boxes for the fields you want to view.



### TIP

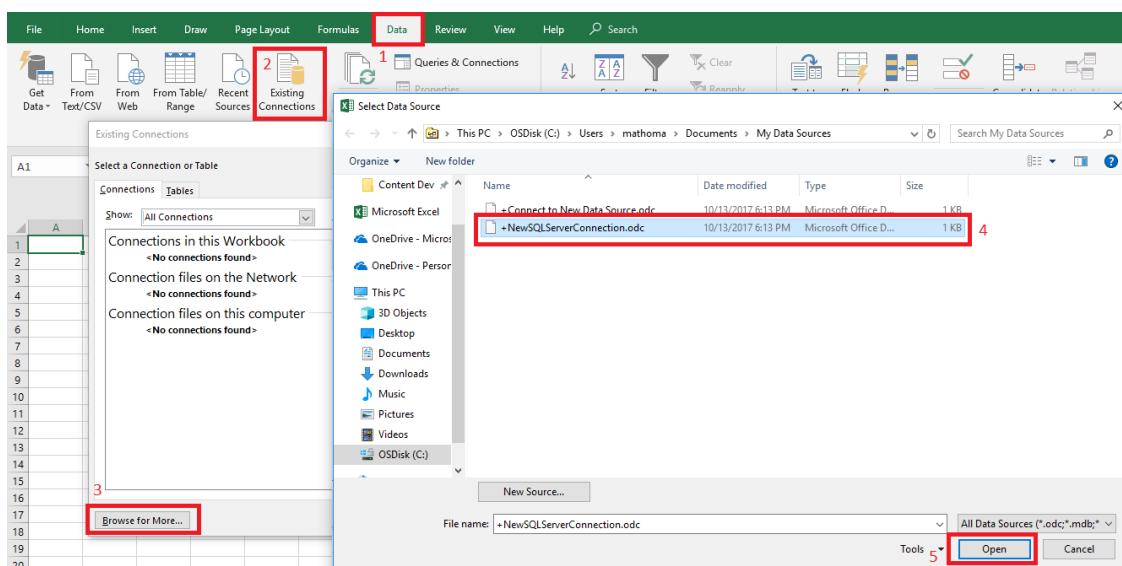
If you want to connect other Excel workbooks and worksheets to the database, select the **Data** tab, and select **Recent Sources** to launch the **Recent Sources** dialog box. From there, choose the connection you created from the list, and then



## 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.

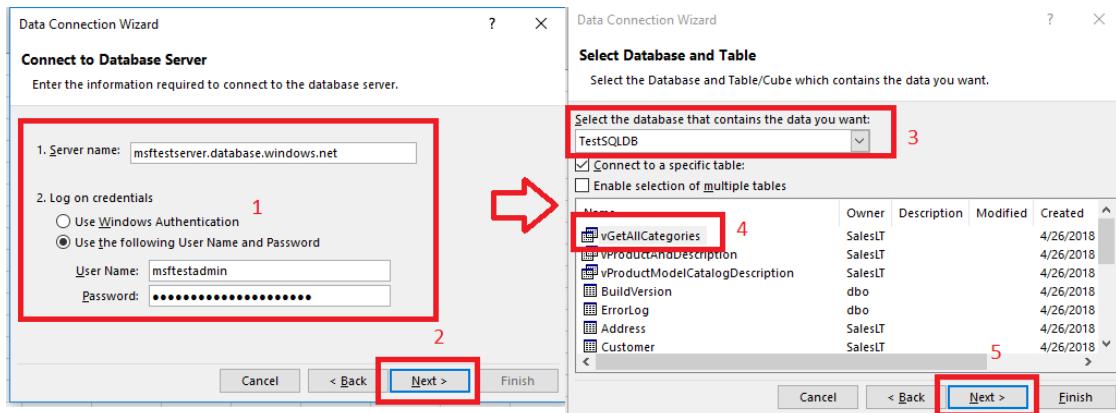
1. In the menu bar at the top of the page, select the **Data** tab, and then select **Existing Connections** to launch the **Existing Connections** dialog box.
  - a. Select **Browse for more** to open the **Select Data Source** dialog box.
  - b. Select the **+NewSqlServerConnection.odc** file and then select **Open** to open the **Data Connection Wizard**.



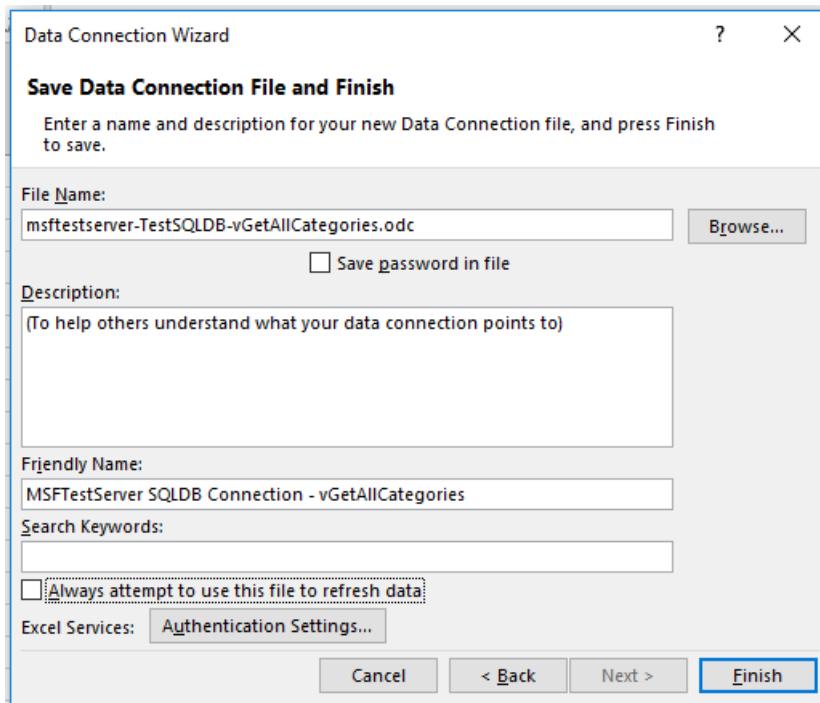
2. In the **Data Connection Wizard**, type in your server name and your SQL Database credentials. Select **Next**.
  - a. Select the database that contains your data from the drop-down.

b. Select the table or view you're interested in. We chose vGetAllCategories.

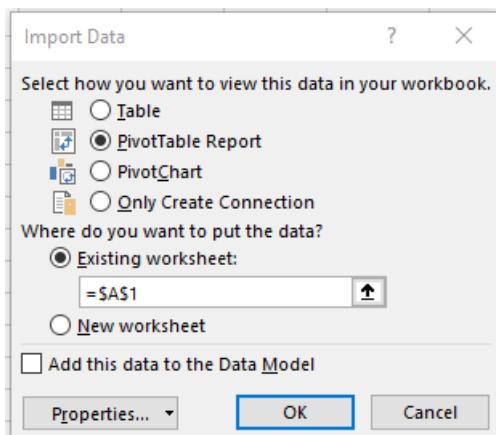
c. Select **Next**.



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.

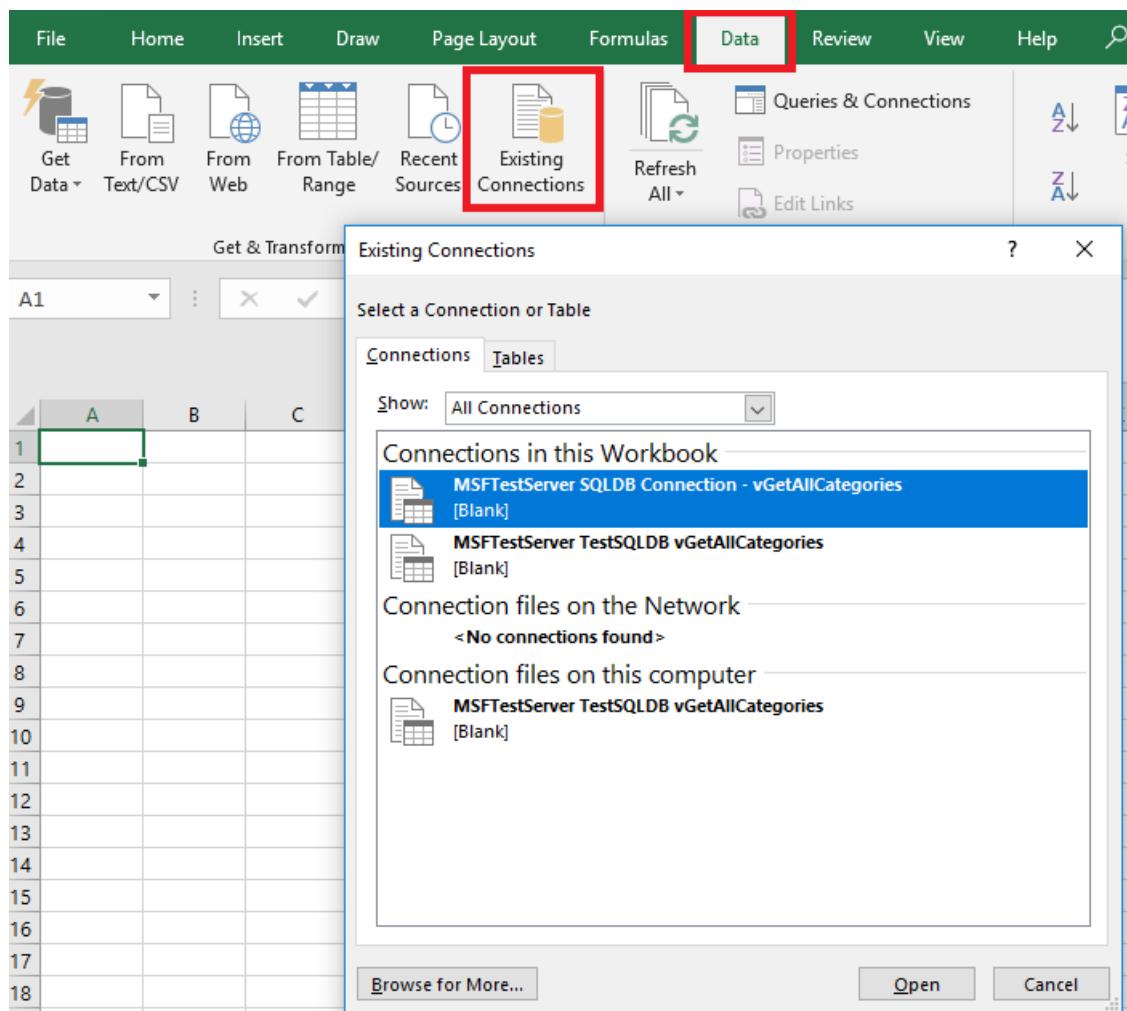


4. Select how you want to import your data. We chose to do a PivotTable. You can also modify the properties of the connection by select **Properties**. Select **OK** when ready. If you did not choose to save the password with the file, then you will be prompted to enter your credentials.



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](#).

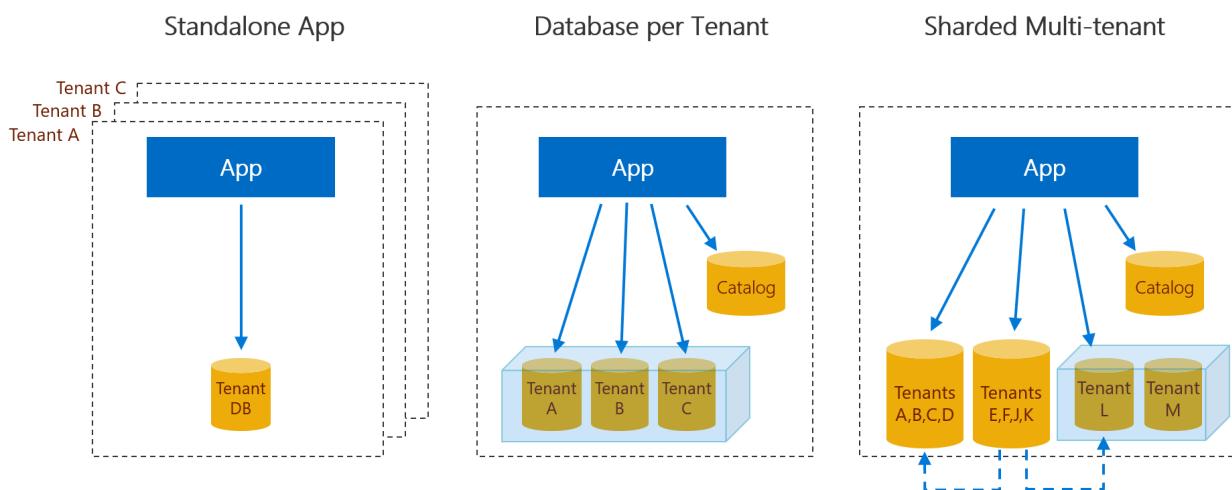
# The Wingtip Tickets SaaS application

9/24/2018 • 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](#).

## 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 a single 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

9/24/2018 • 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

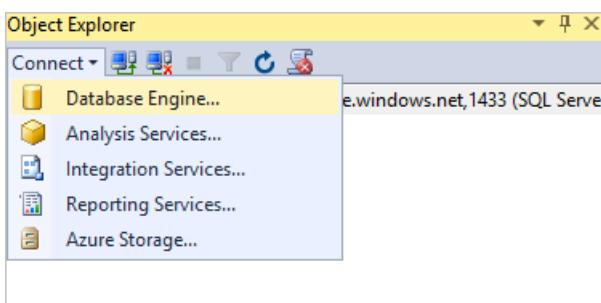
Use [SQL Server Management Studio \(SSMS\)](#) to connect and browse the application servers and databases.

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

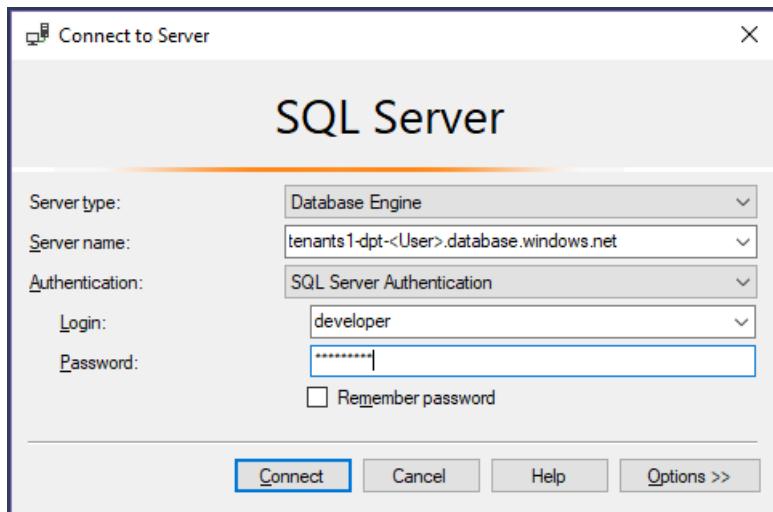
To ensure a successful demo connection, all servers have a [firewall rule](#) allowing all IPs through.

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.

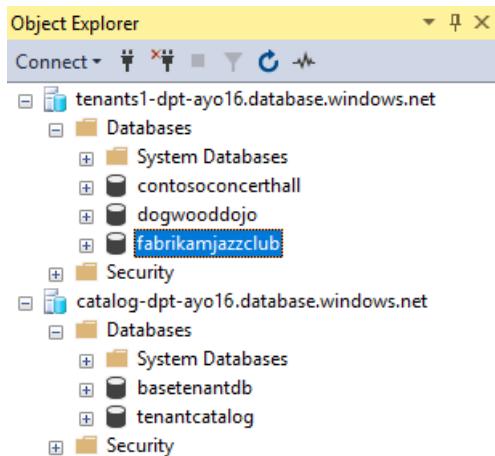


4. Repeat steps 2-3 and connect to the catalog server (see below for specific server names based on the database tenancy pattern selected)

- **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

9/28/2018 • 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 will 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 does not successfully open and compile in Visual Studio 2017 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 will 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 is 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>, substitute the venue name with no spaces. For <user>, substitute 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.

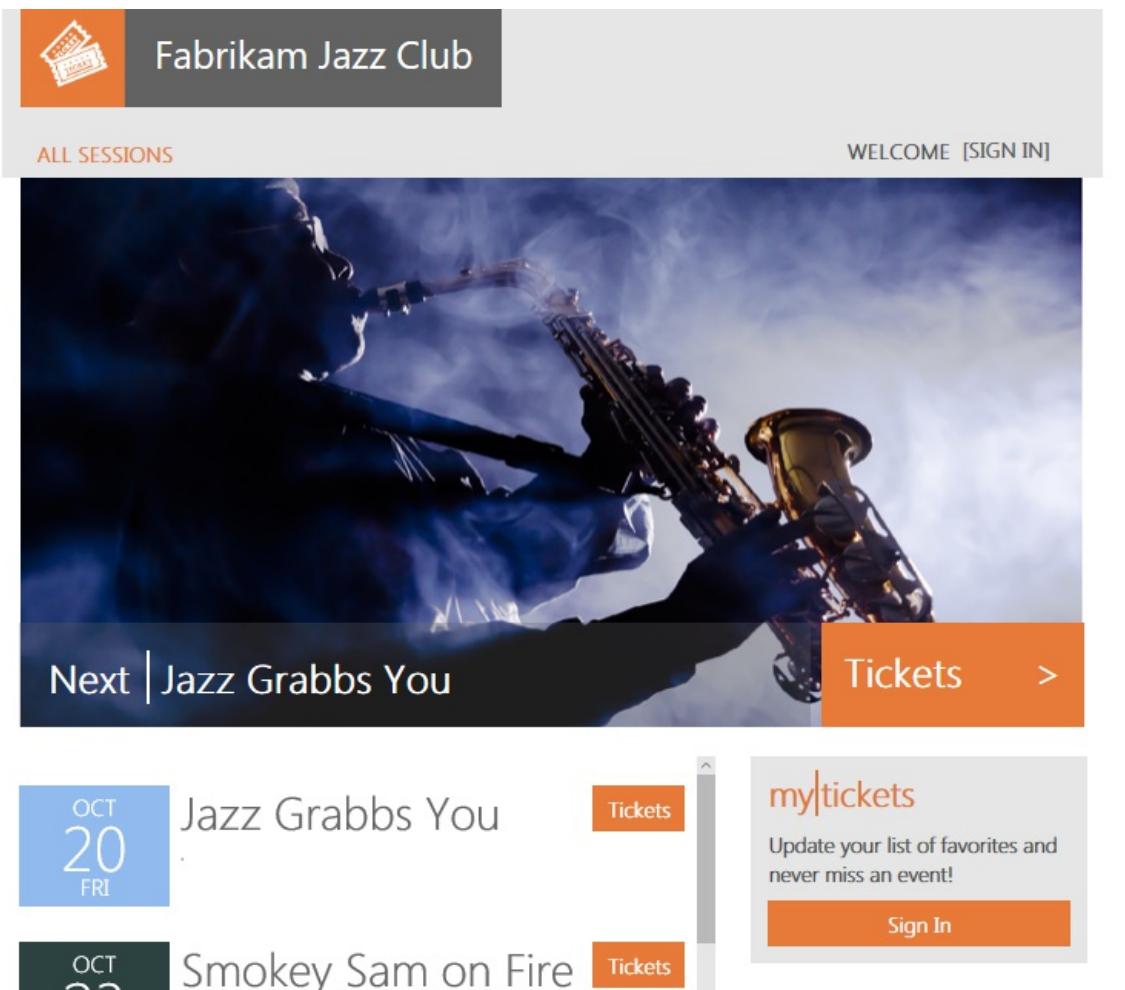
## Run the applications

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 will 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

9/24/2018 • 7 minutes to read • [Edit Online](#)

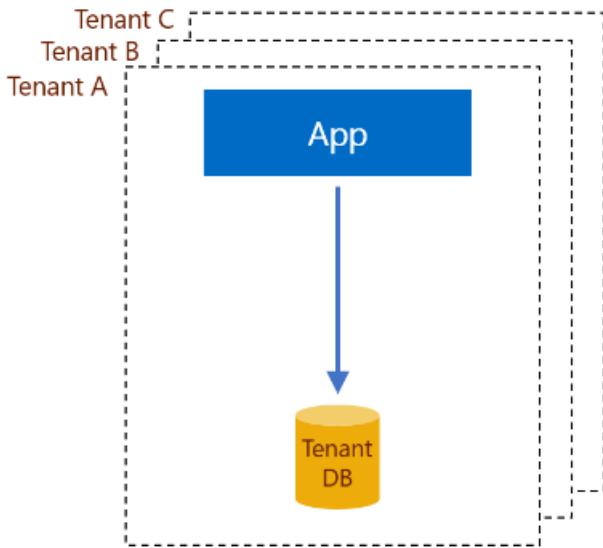
This article covers the provisioning and cataloging of new tenants using the standalone app per tenant SaaS pattern. This article has two major parts:

- 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



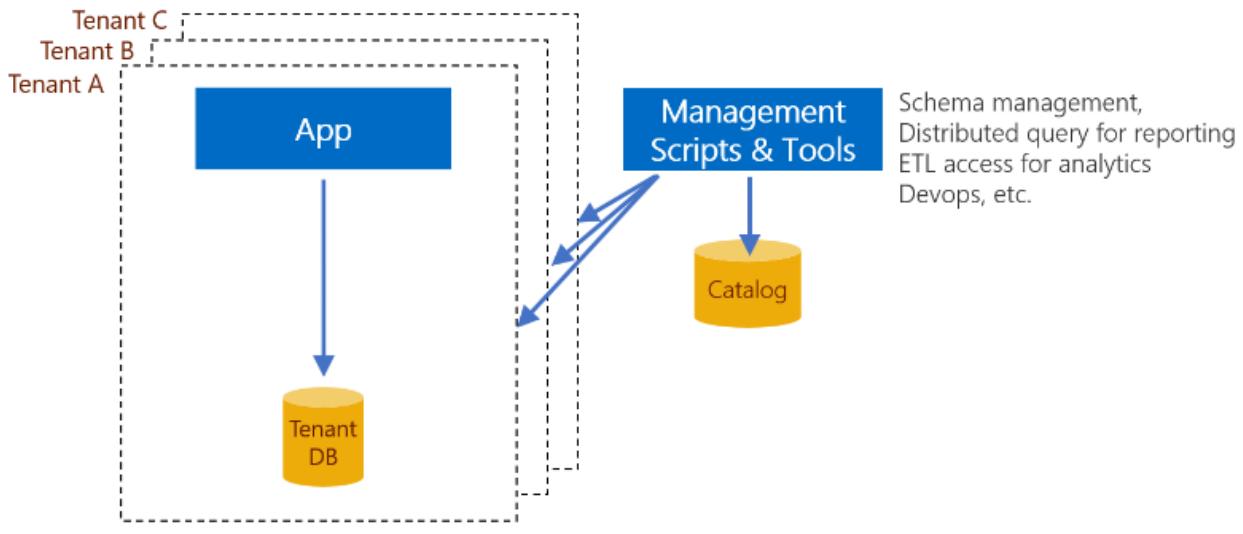
When deploying an application for a tenant, the app and database are provisioned in a new resource group created for the tenant. Using separate resource groups isolates each tenant's application resources and allows them to be managed independently. Within each resource group, each application instance 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.

## Standalone app per tenant



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:

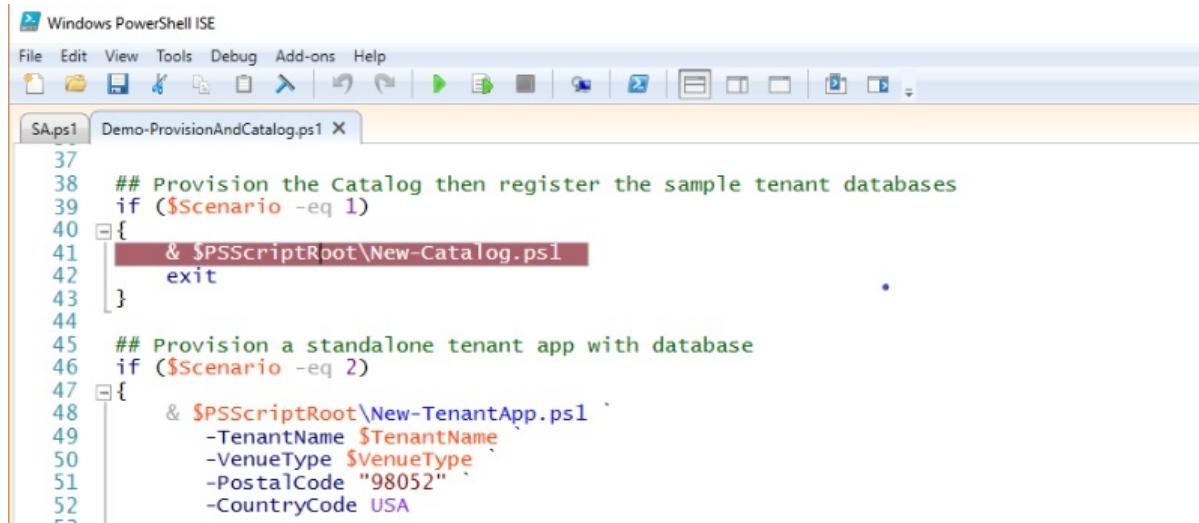
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- The three sample tenant apps are deployed. To deploy these apps in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS Standalone Application pattern](#).

## 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
SA.ps1 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`.

The screenshot shows the Azure portal interface for a SQL database named 'tenantcatalog'. The left sidebar has a 'Data explorer (preview)' section selected. The main area shows the database schema with tables like \_\_ShardManagement.ShardMapManager, \_\_ShardManagement.ShardMappingGL, \_\_ShardManagement.OperationsLogGL, \_\_ShardManagement.ShardMapsGlobal, \_\_ShardManagement.ShardsGlobal, \_\_ShardManagement.ShardedDatabase, and dbo.Tenants. A view named sys.database\_firewall\_rules is also listed. The dbo.TenantsExtended view is selected. A query window titled 'Query 1' contains the SQL command: '1 select \* from dbo.TenantsExtended'. The results pane shows the following data:

| TENANTNAME           | SERVICEPLAN | SERVERNAME                                  | DATABASENAME       |
|----------------------|-------------|---|--------------------|
| Contoso Concert Hall | standard    | contosoconehmerall-bq1.database.windows.net | contosoconehmerall |
| 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 Formula 1 race car in motion, with a 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, there are navigation links: "ALL RACES" on the left and "Tickets" with a right-pointing arrow on the right. At the bottom, there are two event cards: one for "Event 3" on February 5, Monday, and another for "Event 4" on February 6. Each event card has a "Tickets" button. To the right of the events is a sidebar with the heading "my|tickets" and the text "Update your list of favorites and never miss an event!" followed by 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 the 'wingtip-sa-redmapleracing-bg1' resource group. The left sidebar contains a navigation menu with various options. The main content area displays the resource group details, including the subscription information ('Wingtip SaaS - Microsoft Azure Sponsorship'), deployment status ('1 Succeeded'), and a list of resources. The list includes five items: 'events-redmapleracing-bg1' (Traffic Manager profile, global), 'events-redmapleracing-bg1' (App Service plan, South Central US), 'events-redmapleracing-bg1' (App Service, South Central US), 'redmapleracing-bg1' (SQL server, South Central US), and 'redmapleracing' (SQL database, South Central US). There are also filter and grouping options at the top of the list.

## 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

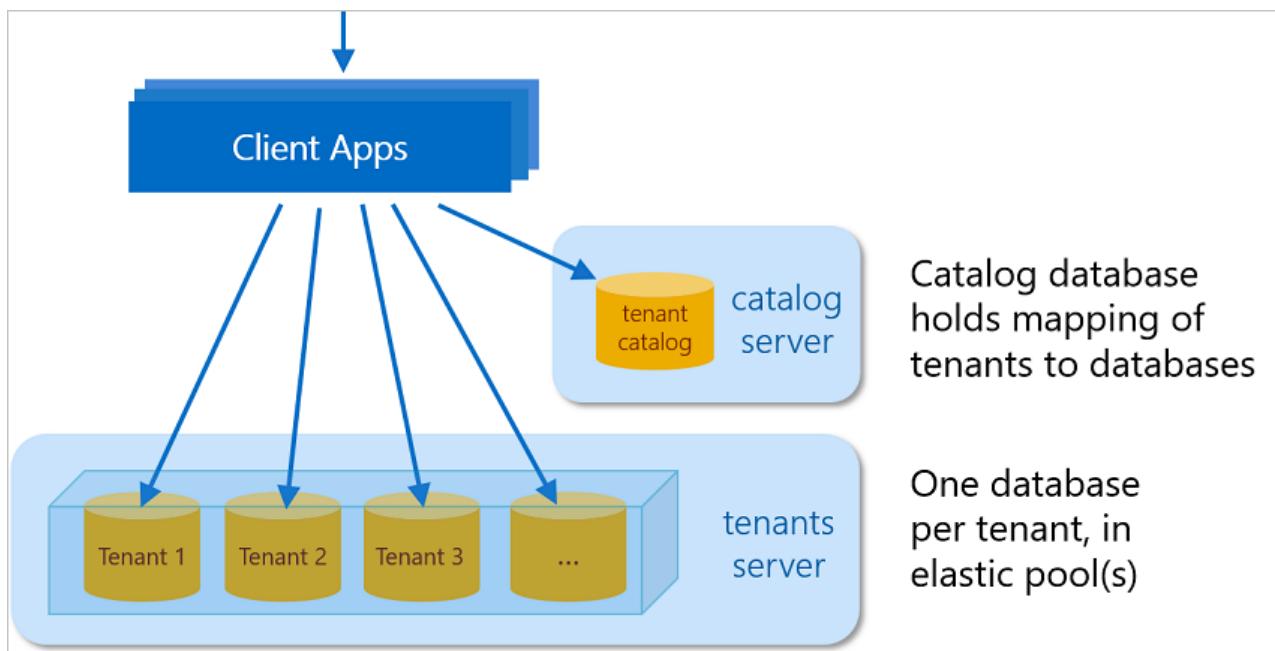
9/24/2018 • 2 minutes to read • [Edit Online](#)

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 Log Analytics</a>                                    | Learn how to use <a href="#">Log Analytics</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

10/29/2018 • 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**.

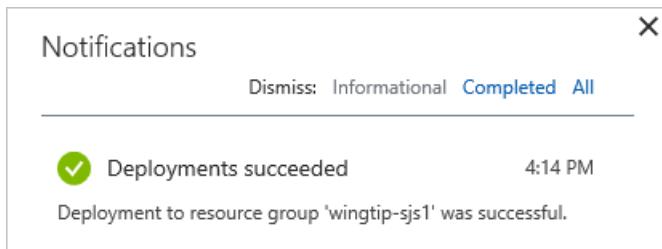


2. Enter values in the template for the required parameters.

**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 'Venues' dropdown menu. Below that, a list of venues is displayed: Contoso Concert Hall, Dogwood Dojo, and Fabrikam Jazz Club. At the bottom right of the list, there's a '1 | End' indicator. The footer contains a link to 'LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE' and information about running on Azure 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

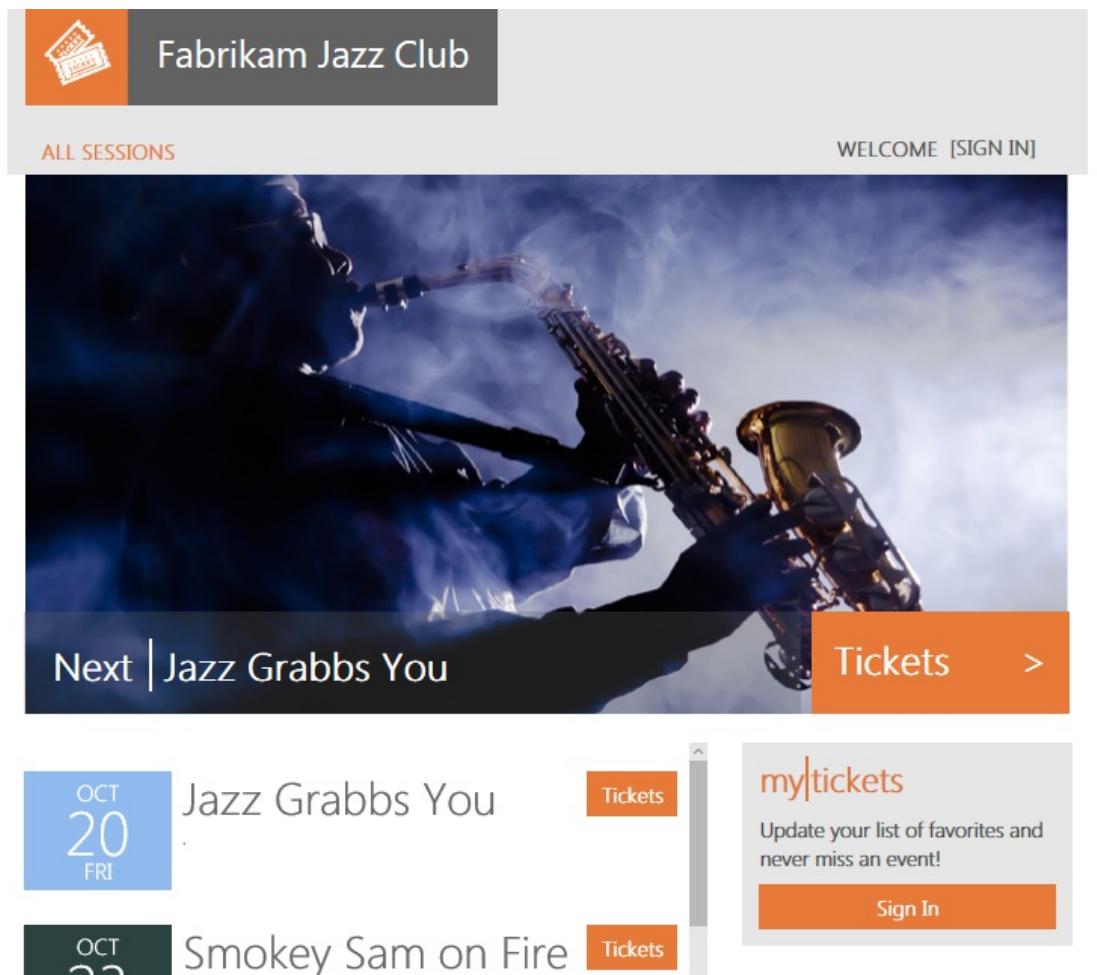
LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE

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.



## 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.<br><br>-dpt distinguishes the <i>database-per-tenant</i> implementation of Wingtip Tickets from other implementations. Examples are the <i>standalone</i> app-per-tenant (-sa) or <i>multitenant database</i> (-mt) implementations. |
| .<user>   | af1 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.



ALL RACES

WELCOME [SIGN IN]



Next | Event 3

Tickets &gt;

MAY  
7  
SUN

Event 3

Performer 3

Tickets

MAY  
10

Event 4

Tickets

my|tickets

Update your list of favorites and  
never miss an event!

Sign In

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 SQL server

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

New database New pool Import database Reset password Move Delete

Essentials

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 |
|--|---------------|--------|--------------|
|  | basetenantdb  | Online | Standard: S1 |
|  | 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.

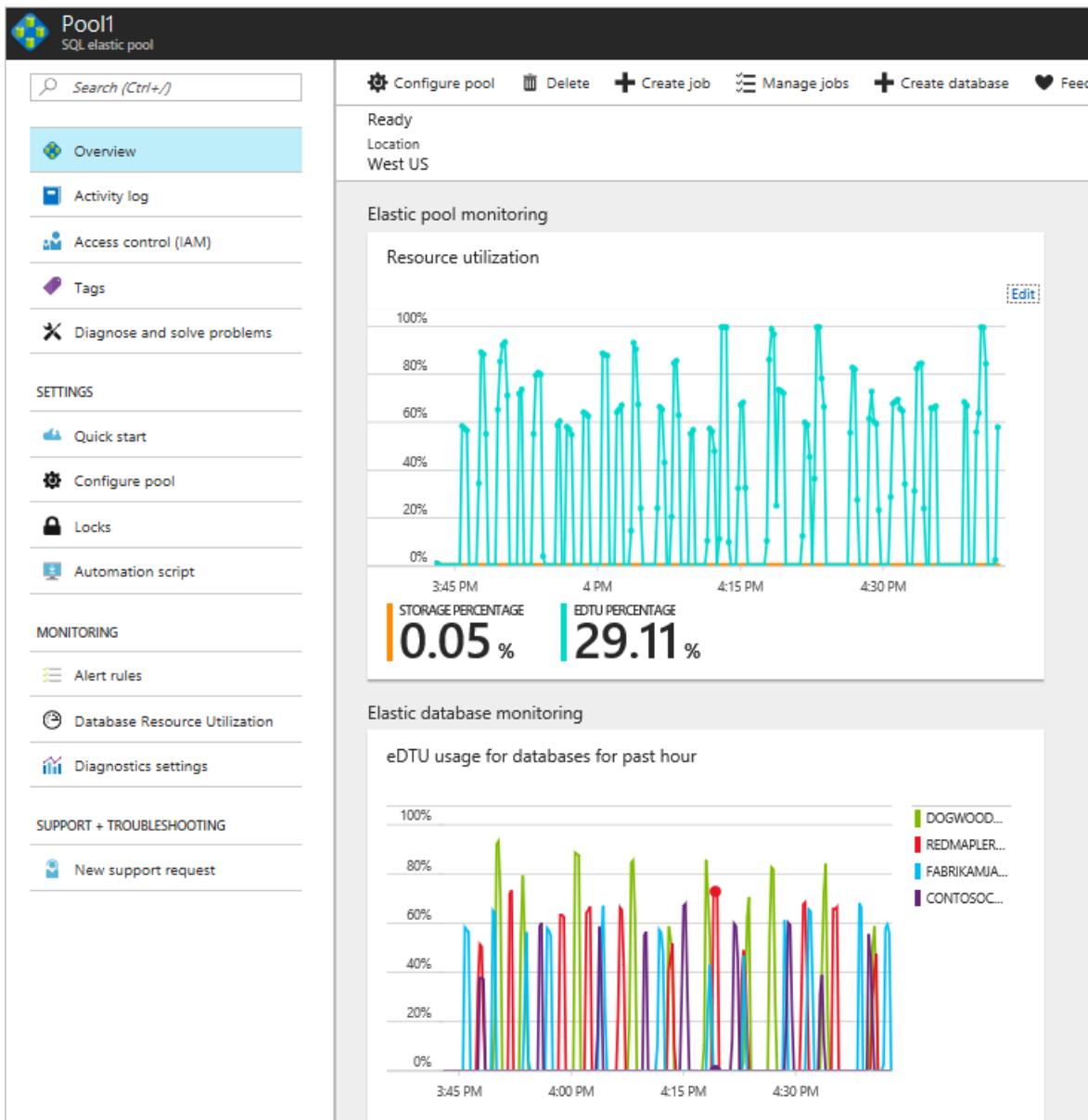
The screenshot shows the Microsoft Azure portal interface for managing SQL databases. The left sidebar lists various management options like Overview, Activity log, and Diagnose and solve problems. The main content area displays the 'Databases' section under 'SQL databases', showing four databases: contosoconcerthall, 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 in the Standard Pool tier with 50 EDUs. Two specific sections are highlighted with red boxes: the list of databases and the details of the single elastic database pool.

| NAME  | PRICING TIER  | POOL EDU |
|-------|---------------|----------|
| 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 standalone 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

9/24/2018 • 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 unlock 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.

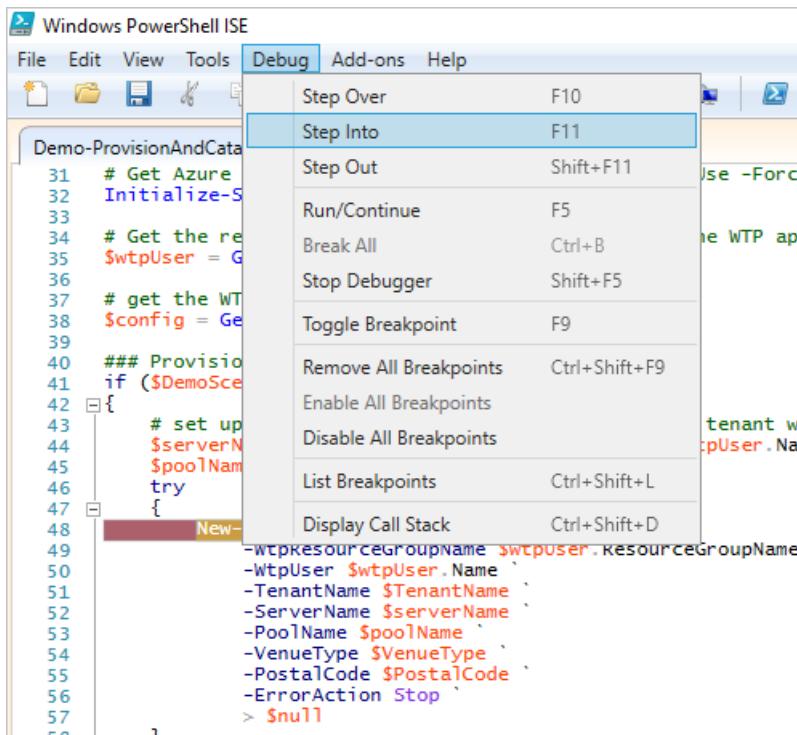
1. In the PowerShell ISE, open ...\\Learning Modules\\ProvisionAndCatalog\\Demo-ProvisionAndCatalog.ps1 and set the following parameters:
  - **\$TenantName** = the name of the new venue (for example, *Bushwillow Blues*).
  - **\$VenueType** = one of the predefined venue types: *blues, classicalmusic, 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      }
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

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.

The screenshot shows the Azure portal interface for managing a SQL server named 'tenants1-'. On the left, there's a sidebar with various icons for monitoring, logs, access control, tags, and troubleshooting. The main content area has a header with 'Report a bug' and 'Search resources'. Below the header, there are buttons for 'New database', 'New pool', 'Import database', 'Reset password', 'Move', and 'Delete'. The 'Essentials' section includes fields for 'Resource group', 'Status' (Available), 'Location' (West US 2), 'Subscription name', 'SQL DB Content', and 'Subscription ID'. The 'Auditing' and 'Server admin' sections show 'Not configured'. The 'Firewall' section has a link to 'Show firewall settings'. The 'Databases' section is titled 'SQL databases' and contains a summary '21 Databases'. A table lists the databases with columns for 'DATABASE', 'STATUS', and 'PRICING TIER'. All databases listed are 'Online' and belong to the 'Elastic Standard' tier. A 'See more' link is at the bottom of the table.

| DATABASE              | STATUS | PRICING TIER     |
|-----------------------|--------|------------------|
| papayaplayers         | Online | Elastic Standard |
| cottonwoodconcerthall | Online | Elastic Standard |
| dogwooddojo           | Online | Elastic Standard |
| osageopera            | Online | Elastic Standard |
| blueoakjazzclub       | Online | Elastic Standard |
| limetreetract         | Online | Elastic Standard |
| juniperjammersjazz    | Online | Elastic Standard |

## 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

10/16/2018 • 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 database 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:

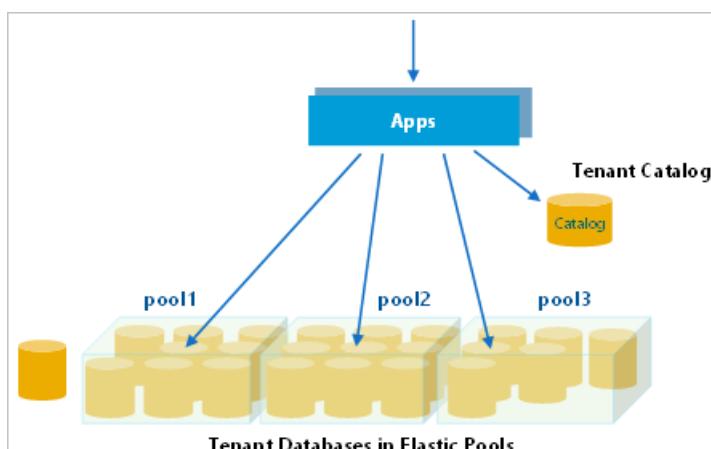
- Simulate usage on the tenant databases by running a provided load generator
- Monitor the tenant databases as they respond to the increase in load
- Scale up the Elastic pool in response to the increased database load
- Provision a second Elastic pool to load balance database activity

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 database 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, [Log Analytics](#) can be used. This is a separate Azure service that provides analytics over emitted diagnostic logs and telemetry gathered in a log analytics workspace. Log Analytics 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 (approx. 40 DTU)                             |
| 3    | Generate load with longer and more frequent bursts per database             |
| 4    | Generate load with higher DTU bursts per database (approx. 80 DTU)          |
| 5    | Generate a normal load plus a high load on a single tenant (approx. 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

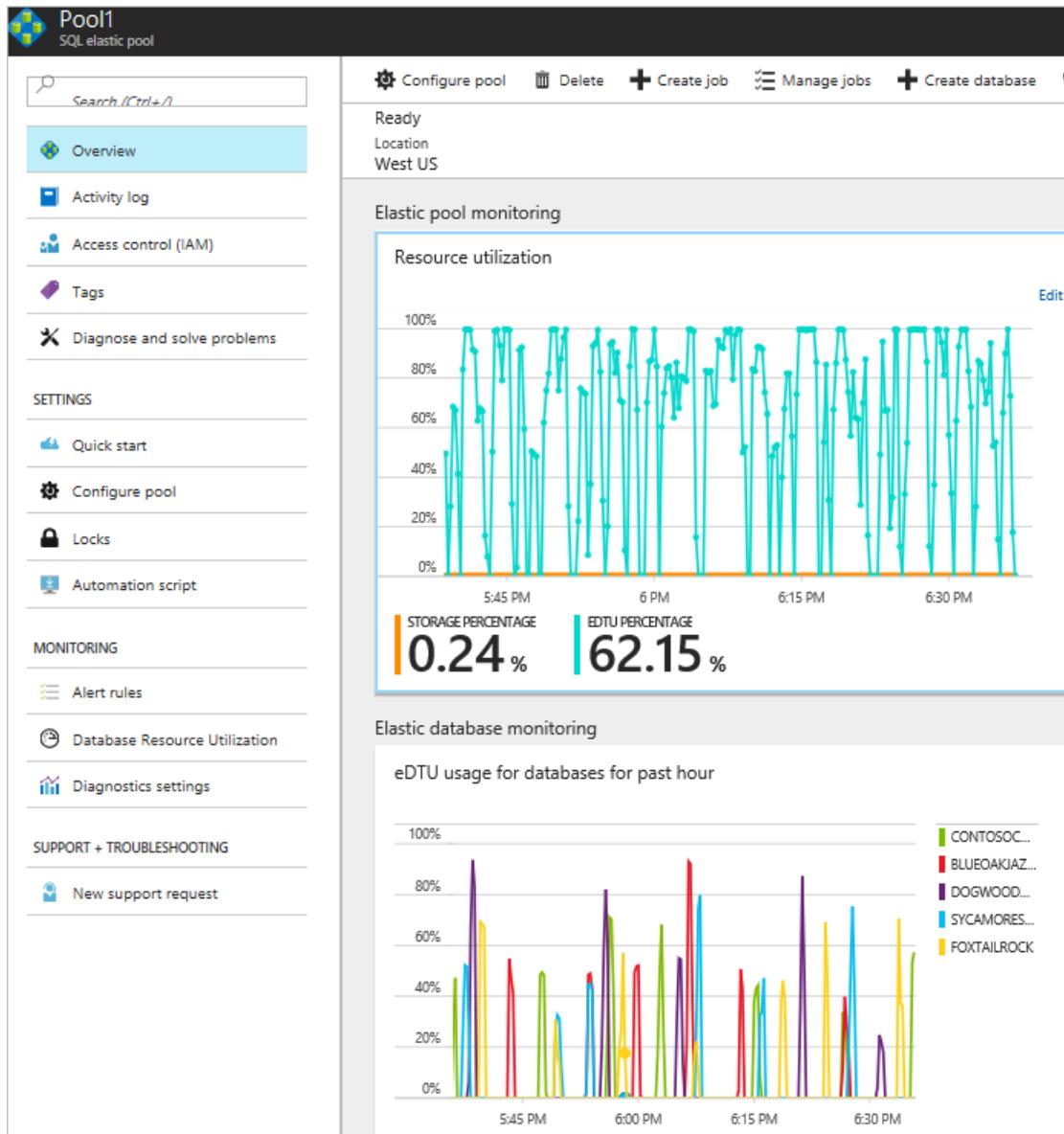
To monitor the resource usage that results from the load being applied, open the portal to the pool containing the tenant databases:

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 |
|--|-------|----------|-----------|
| <input checked="" type="checkbox"/> hornbeamhiphop     | 4.32  | 93.65    |           |
| <input checked="" type="checkbox"/> blueoakjazzclub    | 3.848 | 88.26    |           |
| <input checked="" type="checkbox"/> papayaplayers      | 3.636 | 75.28    |           |
| <input checked="" type="checkbox"/> juniperjammersjazz | 3.321 | 85.89    |           |
| <input checked="" type="checkbox"/> fabrikamjazzclub   | 3.198 | 84.09    |           |
| <input type="checkbox"/> redmapleracing                | 3.091 | 62.4     |           |
| <input type="checkbox"/> mahoganysoccer                | 3.036 | 65.61    |           |
| <input type="checkbox"/> magnoliamotorracing           | 2.911 | 70.3     |           |
| <input type="checkbox"/> contosoconcerthall            | 2.889 | 71.51    |           |
| <input type="checkbox"/> dogwooddojo                   | 2.731 | 57.49    |           |
| <input type="checkbox"/> 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

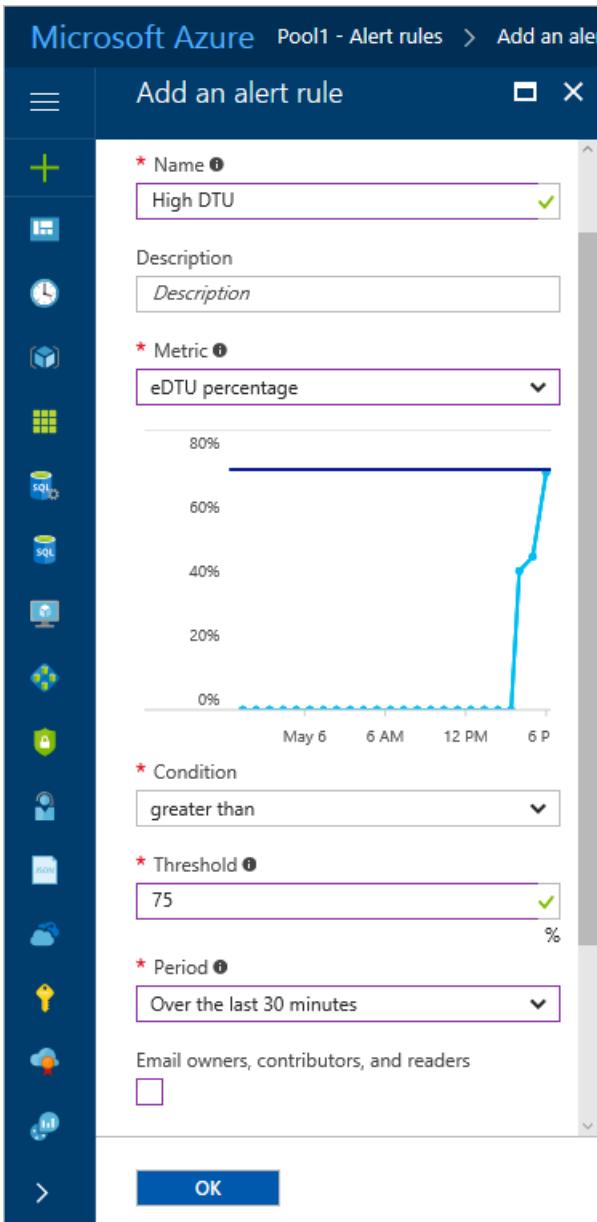
MONITORING

**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.

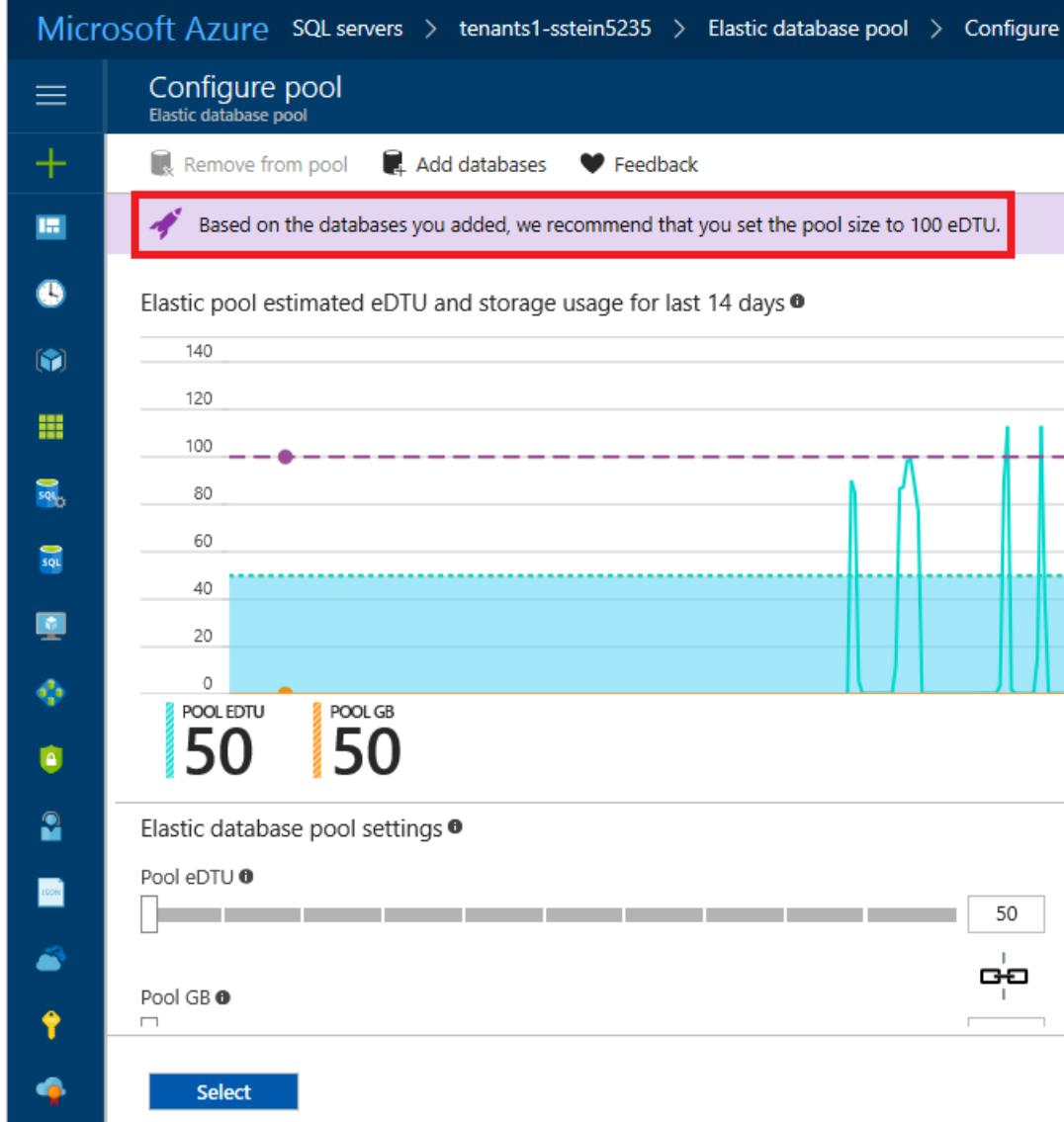
Go back to **Pool1 > Overview** to view the monitoring charts. Monitor the effect of providing the pool with more resources (although with few databases and a randomized load it's not always easy to see conclusively until you run for some time). While you are looking at the charts bear in mind that 100% 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 database 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 a single database

If a single 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 (approx. 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 database 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](#)
- [Log Analytics - Setting up and using Log Analytics tutorial](#)

# Set up and use Log Analytics with a multitenant SQL Database SaaS app

10/18/2018 • 5 minutes to read • [Edit Online](#)

In this tutorial, you set up and use Azure [Log Analytics](#) to monitor elastic pools and databases. This tutorial builds on the [Performance monitoring and management tutorial](#). It shows how to use Log Analytics to augment the monitoring and alerting provided in the Azure portal. Log Analytics supports monitoring thousands of elastic pools and hundreds of thousands of databases. Log Analytics provides a single monitoring solution, which can integrate monitoring of different applications and Azure services across multiple Azure subscriptions.

In this tutorial you learn how to:

- Install and configure Log Analytics.
- Use Log Analytics 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 Log Analytics

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 Log Analytics for monitoring and alerting. Log Analytics is a separate Azure service that enables analytics over diagnostic logs and telemetry that's gathered in a workspace from potentially many services. Log Analytics 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. Log Analytics 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 (approx. 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 and the Azure SQL Analytics solution

Log Analytics is a separate service that must be configured. Log Analytics 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.

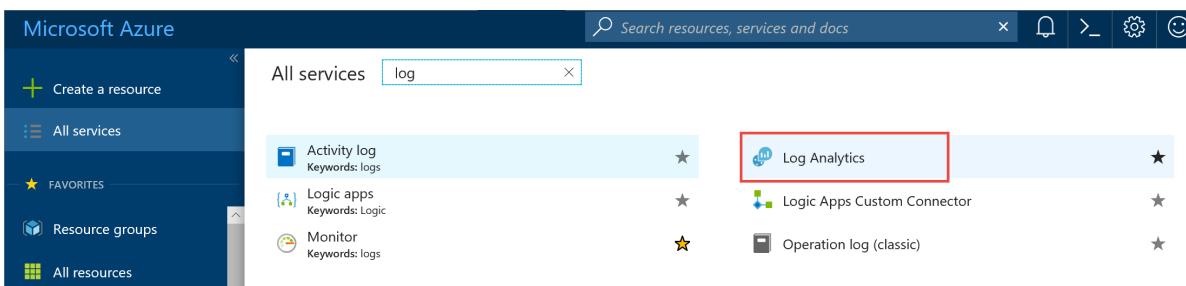
1. In the PowerShell ISE, open ..\WingtipTicketsSaaS-MultiTenantDb-master\Learning Modules\Performance Monitoring and Management\Log Analytics\Demo-LogAnalytics.ps1.
2. To run the script, press F5.

Now you can open Log Analytics 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 and the SQL Analytics solution to monitor pools and databases

In this exercise, open Log Analytics 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. 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.

Microsoft Azure wtplogalytics-sstein5

wtplogalytics-sstein5  
Log Analytics

Search (Ctrl+ /)

OMS Workspace

- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

SETTINGS

- Locks
- Automation script

GENERAL

- Quick Start
- Overview
- Saved searches
- Log Search
- Solutions
- Pricing tier

OMS Portal Delete

Essentials ^

Resource group (change)  
**sstein5**

Status  
Active

Location  
West Central US

Subscription name (change)  
**SQL DB Content**

Subscription ID

Management

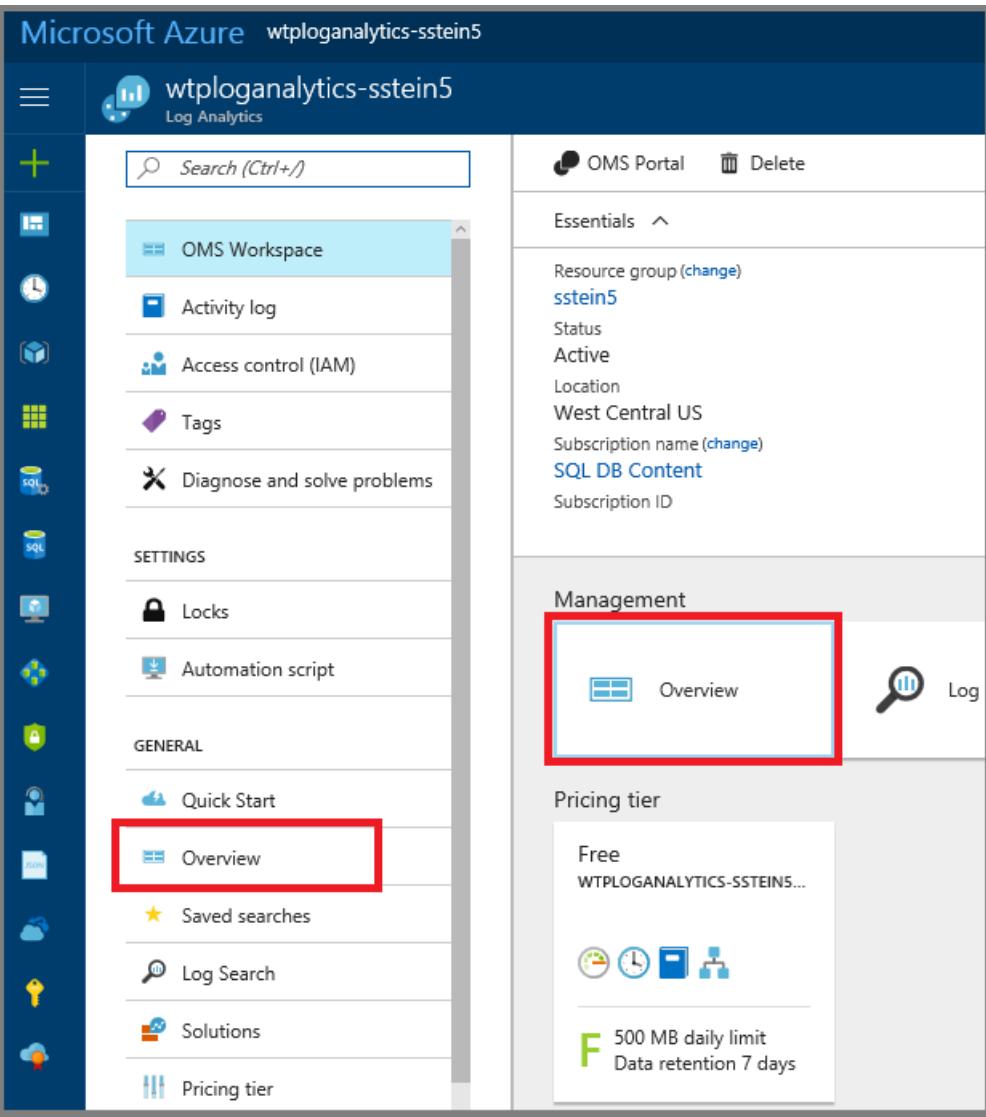
Overview

Log

Pricing tier

Free  
WTPLOGANALYTICS-SSTEINS...

500 MB daily limit  
Data retention 7 days



**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 navigation bar with various icons. The main area displays a summary for 'Azure SQL Analytics (Preview)'. It features a large red box around the following statistics:

- 23** Total SQL Azure Databases
- 1** Total SQL Azure Elastic Pools

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.

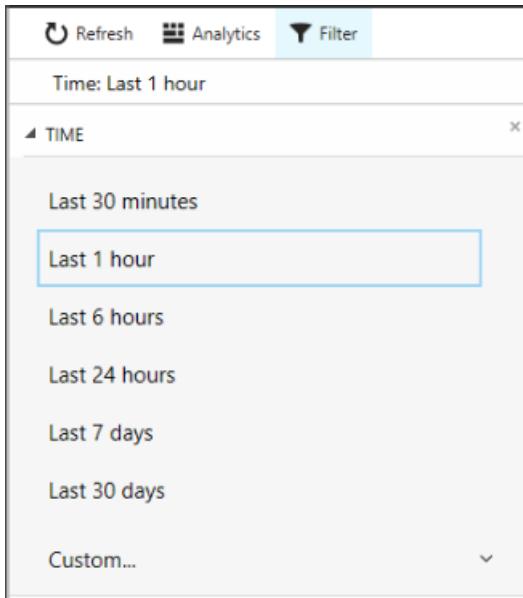
This screenshot shows the 'Azure SQL Analytics (Preview)' summary page. At the top, it displays a table of database metrics:

| DATABASE                          | METRIC | VALUE |
|-----------------------------------|--------|-------|
| tenants1-dpt-bg1.balsamblues...   | CPU    | 8.72  |
| tenants1-dpt-bg1.limetreetrack    | CPU    | 8.25  |
| tenants1-dpt-bg1.mahoganyso...    | CPU    | 7.39  |
| tenants1-dpt-bg1.blueoakjazzcl... | CPU    | 7.22  |
| tenants1-dpt-bg1.magnoliemo...    | CPU    | 6.57  |
| tenants1-dpt-bg1.foxtailrock      | CPU    | 6.25  |
| tenants1-dpt-bg1.sycamoresym...   | CPU    | 5.78  |
| catalog-dpt-bg1.tenantcatalog     | CPU    | 5.33  |
| tenants1-dpt-bg1.mangrovesoc...   | CPU    | 5.33  |
| tenants1-dpt-bg1.cottonwoodc...   | CPU    | 5.14  |
| tenants1-dpt-bg1.dogwooddojo      | CPU    | 4.74  |
| tenants1-dpt-bg1.hornbeamhip...   | CPU    | 4.69  |
| tenants1-dpt-bg1.greenmapler...   | CPU    | 4.45  |
| tenants1-dpt-bg1.tamarindstudio   | CPU    | 4.15  |
| tenants1-dpt-bg1.staranisejudo    | CPU    | 4.07  |

Below this are several performance charts:

- Number of resources per utilization bucket**: A bar chart showing utilization across four buckets (0-5, 5-10, 10-15, 15-20).
- Query duration in seconds**: A line chart showing query duration over time.
- Total time queries spent waiting per type**: A stacked bar chart showing waiting time by query type.
- Total time DBs spent waiting per type**: A stacked bar chart showing waiting time by database type.

7. Change the filter setting to modify the time range. For this tutorial, select **Last 1 hour**.



8. Select a single database to explore the query usage and metrics for that database.

The screenshot shows the Azure SQL Analytics (Preview) page for a database named 'balisambucusclub'. The page includes the following sections:

- RESOURCE INFO:** Subscription name: 112900e0-ad0e-4eb8-aea2-b09a07253f30; Server name: tenantz1-dpt-bg1; Database name: balisambucusclub.
- INSIGHTS SUMMARY:** Requires configuration. Please make sure that **SQLInsights** Diagnostics log is enabled. [Learn more about SQL Diagnostics logs](#).
- QUERIES:** A chart showing Query Wait (ms) over time from 3:00 PM to 4:00 PM. The chart shows a peak at approximately 3:30 PM. Below the chart is a table of query metrics:

| QUERY              | METRIC   | MAX (s) | Avg (%) | Dominant Wait | Max (s) | Avg (%) | EXCS |
|--------------------|----------|---------|---------|---------------|---------|---------|------|
| 0xAB1EF762A66266   | Duration | 0       | 0       | MEMORY        | 0       | 0       | 480  |
| 0xC824056576DF...  | Duration | 0       | 0       |               | 0       | 0       | 6    |
| 0x8500350EAFD...   | Duration | 0       | 0       |               | 0       | 0       | 2    |
| 0x8906E91367EA1... | 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

[Refresh](#) [Analytics](#) [Filter](#)

Time: Last 1 hour

| METRIC                | MAX   | 80TH PERCE... | AVG  |
|-----------------------|-------|---------------|------|
| storage [GB]          | 0.01  | 0.01          | 0    |
| cpu_percent           | 68.14 | 68.14         | 5.57 |
| dtu_consumption...    | 68.14 | 68.14         | 5.57 |
| dtu_limit             | 50    | 50            | 50   |
| dtu_used              | 34.07 | 34.07         | 2.78 |
| workers_percent       | 1.67  | 1.67          | 0.90 |
| connection_succes...  | 1     | 1             | 0.04 |
| sessions_percent      | 0.58  | 0.58          | 0.56 |
| log_write_percent     | 0.04  | 0.04          | 0    |
| physical_data_read... | 0     | 0             | 0    |
| connection_failed     | 0     | 0             | 0    |
| blocked_by_firewall   | 0     | 0             | 0    |
| deadlock              | 0     | 0             | 0    |
| storage_percent       | 0     | 0             | 0    |
| xtp_storage_percent   | 0     | 0             | 0    |

### DATABASE METRICS

10. Scroll the analytics page to the left, and select the server tile in the **Resource Info** list.

RESOURCE INFO

|  |  |
|--|--|
| Subscription name:<br>112900e0-ad0e-4ab8-aea2-b09a07253f30 |  |
| Server name:<br>tenants1-dpt-bg1                           |  |
| Database name:<br>balsambluesclub                          |  |

A page opens that shows the pools and databases on the server.

Server  
wtplogalytics-bg1

Refresh Analytics Filter

Time: Last 1 hour

| RESOURCE INFO  |       |        | ELASTIC POOLS |        |       | DATABASES             |        |       |
|--|-------|--------|---------------|--------|-------|-----------------------|--------|-------|
|  | POOL  | METRIC | POOL          | METRIC | VALUE | DATABASE              | METRIC | VALUE |
| Subscription name:<br>112900e0-ad0e-4ab8-aea2-b09a07253f30 | pool1 | CPU    | pool1         | CPU    | 78.86 | limetreetrack         | CPU    | 8.63  |
| Server name:<br>tenants1-dpt-bg1                           |       |        |               |        |       | balsambluesclub       | CPU    | 8.31  |
|  |       |        |               |        |       | 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 > wtplogalytics-bg1 > Overview > Azure SQL Analytics (Preview) > Database > Server > Elastic Pool

Elastic Pool  
wtplogalytics-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**

The chart displays the usage of elastic database units (eDTUs) over a 30-minute period. The Y-axis represents eDTUs from 0 to 100, and the X-axis shows time from 3:00 PM to 3:30 PM. The data shows significant fluctuations, with peaks around 100 eDTUs and troughs around 20 eDTUs.

**Storage**

The chart displays storage usage over a 30-minute period. The Y-axis represents storage from 0 to 100, and the X-axis shows time from 3:00 PM to 3:30 PM. The data shows a constant value of 0, indicating no storage usage.

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. On the left, there's a sidebar with links like 'OMS Workspace', 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. Below that is a 'SETTINGS' section with 'Locks', 'Automation script', and 'Advanced settings'. The main area has tabs for 'Essentials', 'Management', and 'Metrics'. Under 'Management', there are four buttons: 'Overview' (selected), 'Log Search', 'OMS Portal' (highlighted with a red box), and 'View Designer'. The 'Essentials' tab displays workspace details: Resource group (wingtip-dpt-bg1), Status (Active), Location (East US), Subscription name (Wingtip SaaS - Microsoft Azure Sponsorship), and Pricing tier (Free). It also shows the workspace name (wtploganalytics-bg1), ID (02123338-8485-4021-87e8-fd3e5508f36b), Management services (Operations logs), and Subscription ID.

In the Log Analytics workspace, you can explore the log and metric data further.

Monitoring and alerting in Log Analytics 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 Log Analytics to query and set alerts, see [Work with alert rules in Log Analytics](#).

Log Analytics 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 Log Analytics.
- Use Log Analytics 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 Log Analytics](#)

# Restore a single tenant with a database-per-tenant SaaS application

9/24/2018 • 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 [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 unblock 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'. The table lists ten venue names: Balsam Blues Club, Blue Oak Jazz Club, Bushwillow Blues, Contoso Concert Hall, Cottonwood Concert Hall, Dogwood Dojo, Fabrikam Jazz Club, Foxtail Rock, Hackberry Hitters, and Hornbeam HipHop. The row for 'Contoso Concert Hall' is highlighted with a red rectangle. At the bottom right of the table area, there are 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<br>26<br>FRI | Handel's Messiah<br>Contoso Symphony          |  | Tickets |
| MAY<br>29<br>MON | Moonlight Serenade<br>Contoso Quartet         |  | Tickets |
| JUN<br>1<br>THU  | Seriously Strauss<br>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-sj2.database.windows.net  
database: contosoconcerthall\_old tenant id: 5D172CF1

### "Accidentally" delete the last event

1. 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).**
2. 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.
```

3. 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<br>23<br>TUE     | The 1812 Overture<br>Contoso Symphony | Tickets  |
|----------------------|---------------------------------------|--|
| MAY<br>26<br>FRI     | Handel's Messiah<br>Contoso Symphony  | Tickets  |
| MAY<br>29<br>MON     | Moonlight Serenade<br>Contoso Quartet | Tickets  |
| Contoso Concert Hall |                                       | LEARN HOW TO BUILD A SAAS APP ON SQL DATABASE<br>Running on Azure SQL Database. © 2017 Microsoft<br>server: tenants1-sjs2.database.windows.net<br>database: contosoconcerthall tenant id: F5C9F146 |

POWERED BY THE WINGTIP TICKETS PLATFORM

**my|tickets**  
Update your list of favorites and never miss an event!  
[Sign In](#)

## 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 [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

9/24/2018 • 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:

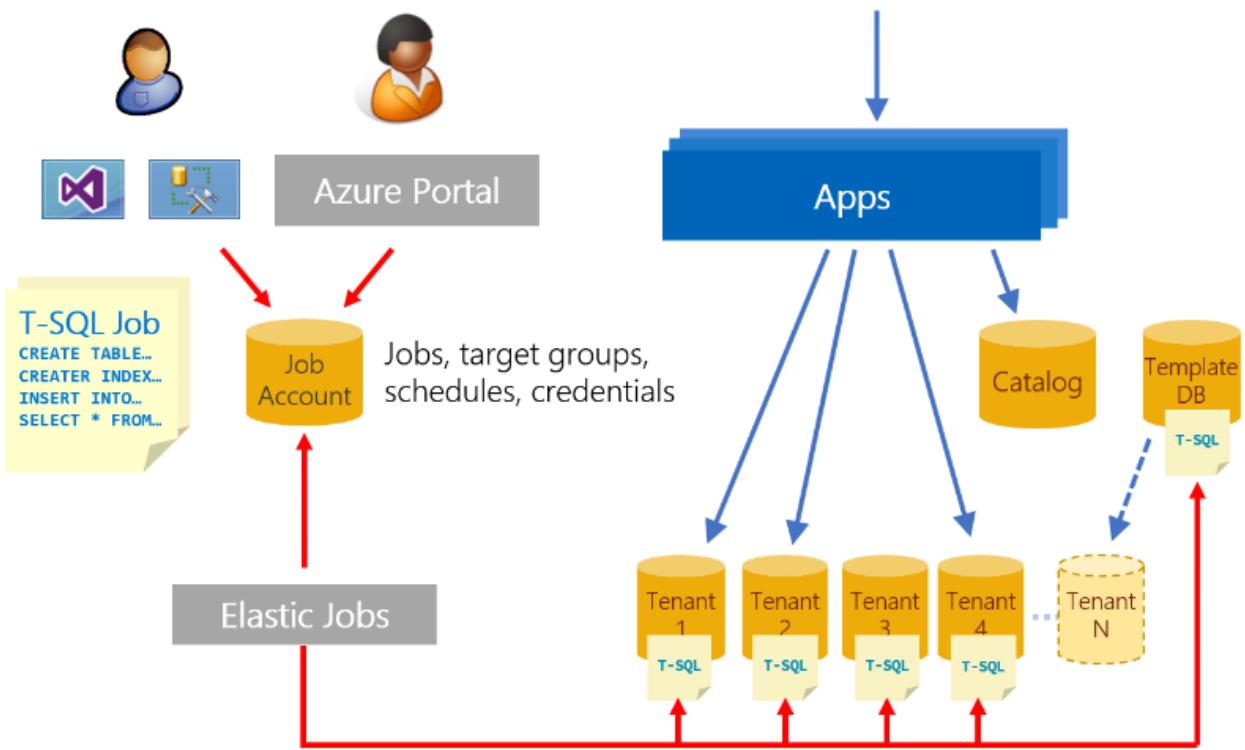
- The Wingtip Tickets SaaS Database Per Tenant app is deployed. To deploy in less than five minutes, see [Deploy and explore the Wingtip Tickets SaaS database per tenant application](#)
- Azure PowerShell is installed. For details, see [Getting started with Azure PowerShell](#)
- The latest version of 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](#)
- [Create and manage scaled-out cloud databases](#)

# Cross-tenant reporting using distributed queries

10/30/2018 • 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.

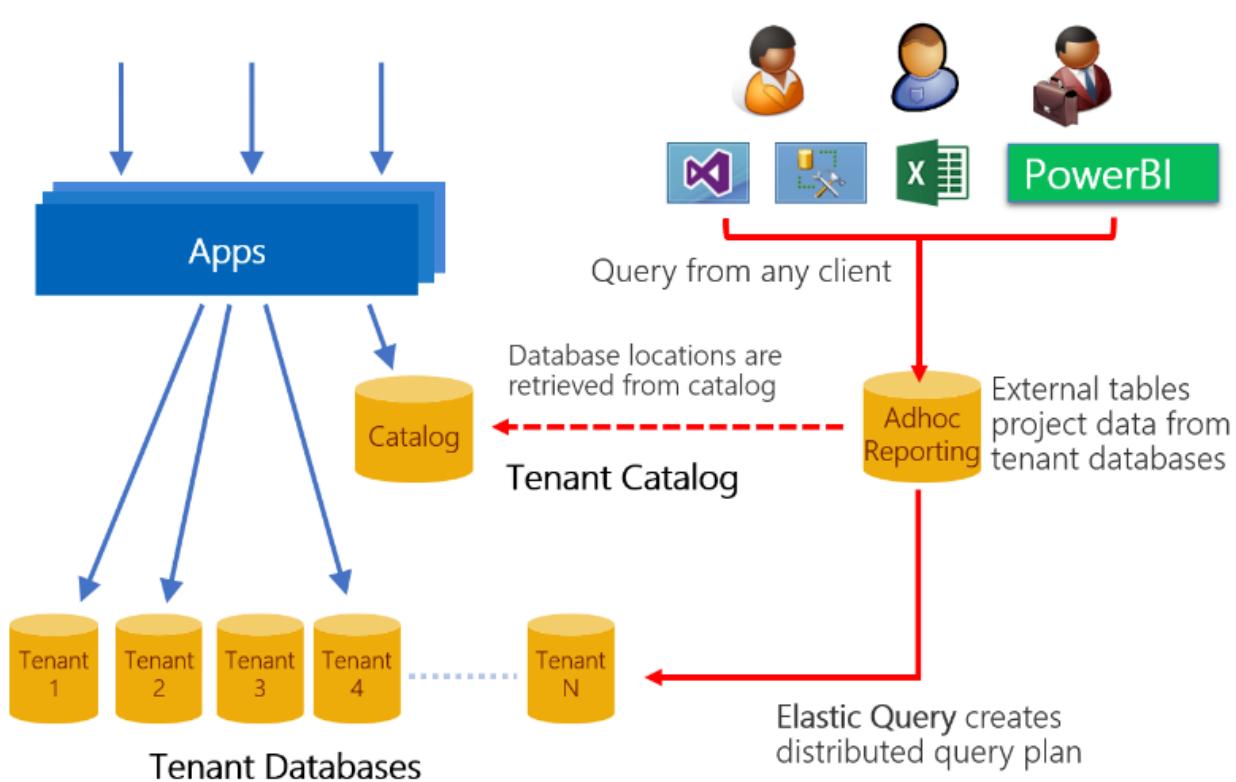
In this tutorial you learn:

- 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

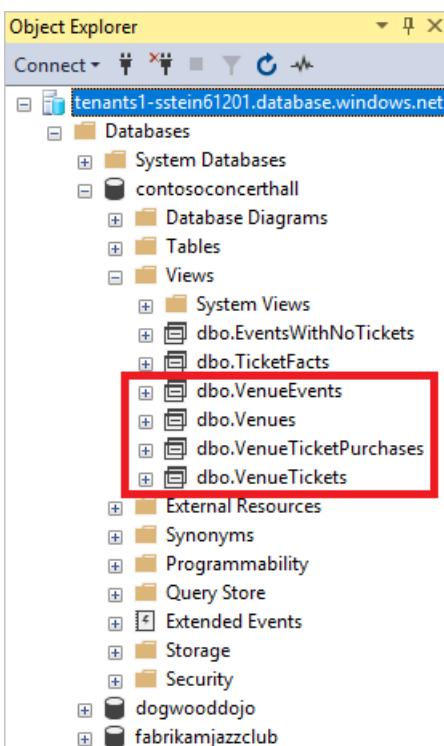
-- The base Events table, which has no VenueId column.
SELECT * FROM Events

-- This view projects the VenueId retrieved from the Venues table.
SELECT * FROM VenueEvents
```

In these views, the *Venuelid* 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 *Venuelid*.

## 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 a single 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.

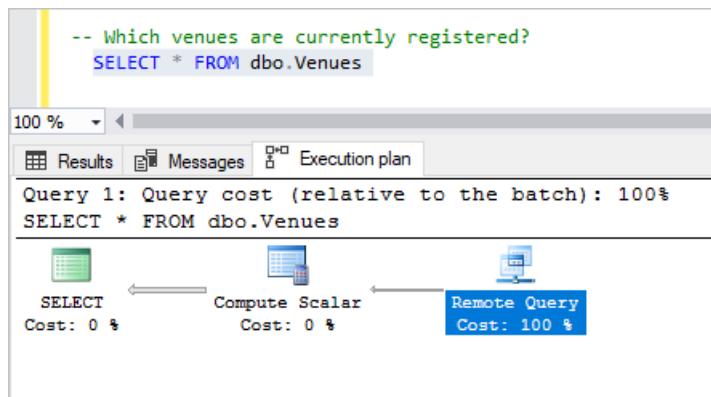
When inspecting the execution plan, hover over the plan icons for details.

Important to note, is that setting **DISTRIBUTION = SHARDED(VenueId)** when the external data source is defined improves performance for many scenarios. As each *VenueId* maps to a single 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
```

Execution plan:

```
00 % ▾
Results Messages Execution plan
Query 1: Query cost (relative to the batch): 100%
SELECT VenueName, VenueTypeName, EventTypeName FROM dbo.Venues INNER JOIN c
[VenueTypes] . [PK_VenueTyp_265E44F...]
Cost: 1 %

SELECT Cost: 0 %
      Hash Match (Inner Join)
      Cost: 7 %

Clustered Index Scan (Clustered)
[VenueTypes] . [PK_VenueTyp_265E44F...]
Cost: 1 %

Compute Scalar Cost: 0 %
Remote Query Cost: 92 %
```

6. 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
```

Execution plan:

```
00 % ▾
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
Cost: 0 %

SELECT Cost: 0 %
      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 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

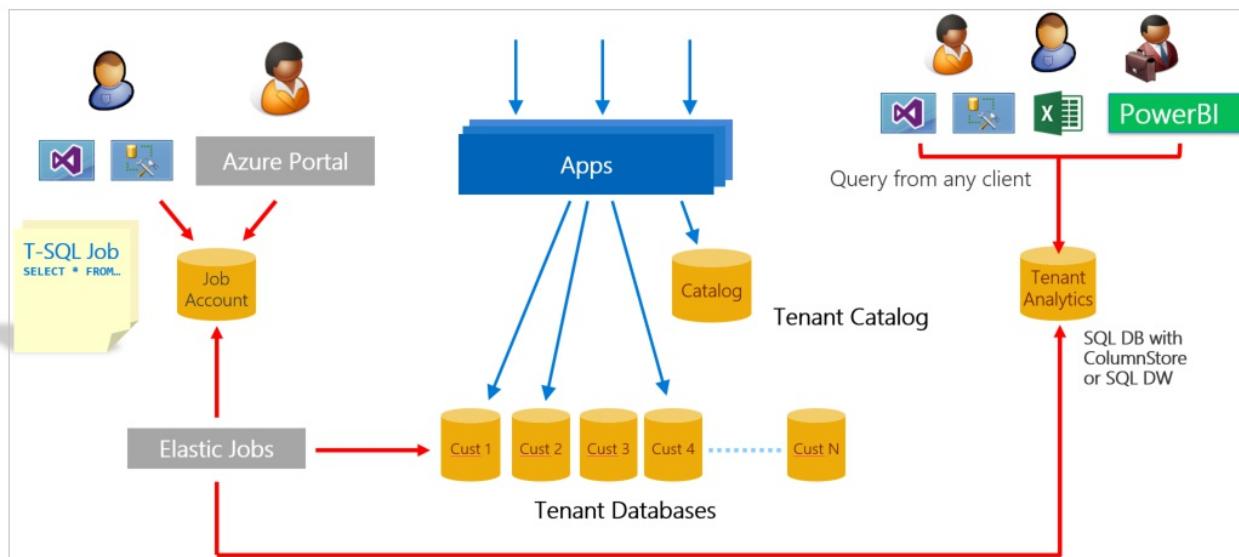
9/26/2018 • 13 minutes to read • [Edit Online](#)

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.

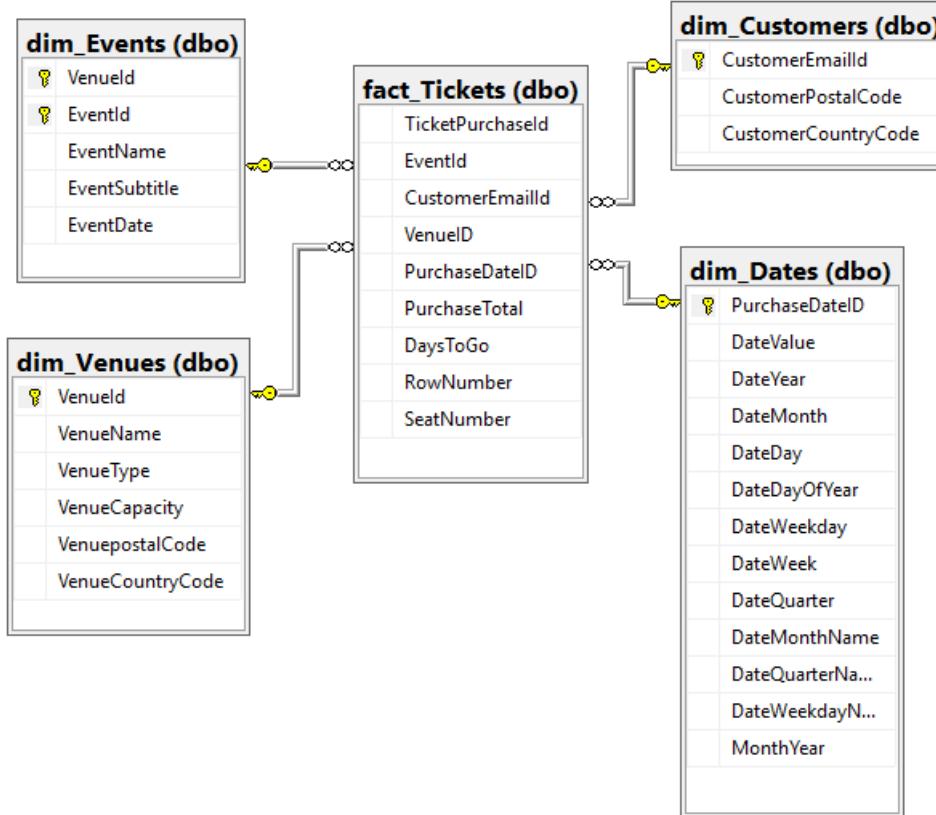
Accessing data for all tenants is simple when all the data is in just one multi-tenant database. But the access is more complex when distributed at scale across potentially thousands of databases. One way to tame the complexity and to minimize the impact of analytics queries on transactional data is 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 *unlock the zip file* before extracting its contents. Check out the [general guidance](#) for steps to download and unlock 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.

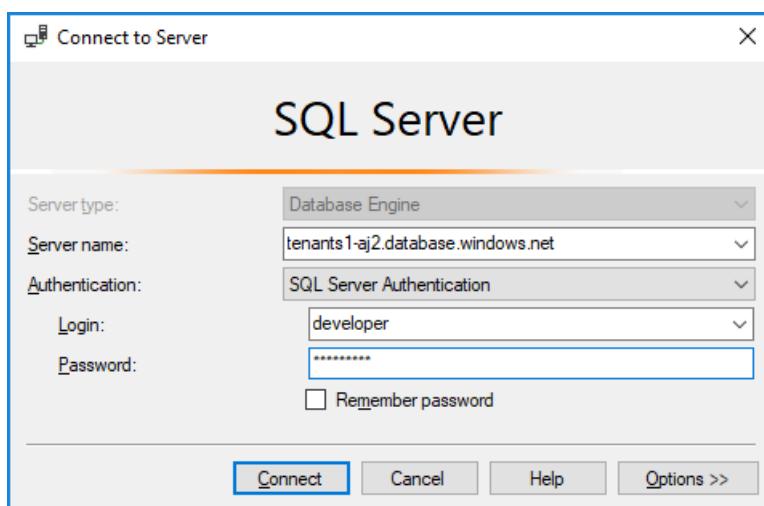
### Deploy the analytics store

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.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.

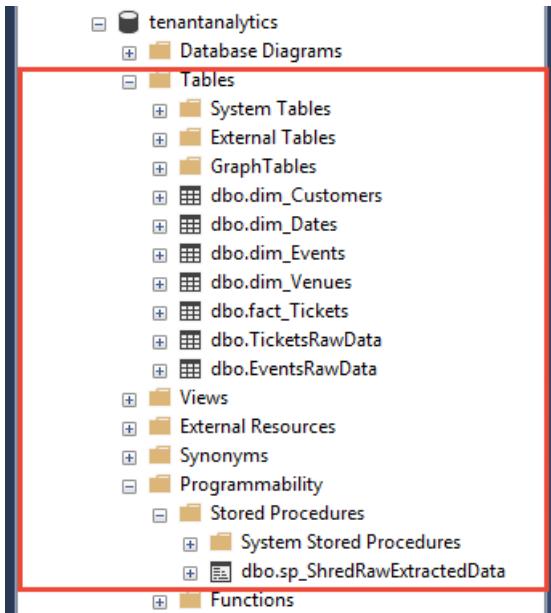


In the Object Explorer, perform the following steps:

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 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 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 tables like dbo.dim\_Customers, dbo.dim\_Dates, and dbo.TicketsRawData. The central pane displays the SQL script `ExtractTickets.sql`, which creates a job named 'ExtractTickets' and runs it against each tenant database. The results grid on the right shows a list of 15 job executions, each with details like job\_id, job\_name, step\_id, and lifecycle status. A message at the bottom indicates the query was executed successfully.

| job_execution_id | job_name                             | job_id          | job_version_number                   | step_id | is_active | lifecycle |
|------------------|--------------------------------------|-----------------|--------------------------------------|---------|-----------|-----------|
| 1                | B6FEFC9-E010-4BE6-AAC5-B201BED399A7  | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 2                | B6FEFC9-E010-4BE6-AAC5-B201BED399A7  | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 3                | 4D893CC9-BF6E-4A20-923F-1764400A6A82 | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 4                | D654DA06-2CE0-40DA-B55D-FB10EE0CF333 | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 5                | 0C85941E-2AA6-4FC2-A757-C9CB1BCE0F   | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 6                | 0C85941E-2AA6-4FC2-A757-C9CB1BCE0F   | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 7                | 0C85941E-2AA6-4FC2-A757-C9CB1BCE0F   | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 8                | 4D893CC9-BF6E-4A20-923F-1764400A6A82 | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | NULL      | 0         |
| 9                | 0C85941E-2AA6-4FC2-A757-C9CB1BCE0F   | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 10               | 4D893CC9-BF6E-4A20-923F-1764400A6A82 | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 11               | D654DA06-2CE0-40DA-B55D-FB10EE0CF333 | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 12               | 4D893CC9-BF6E-4A20-923F-1764400A6A82 | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 13               | B6FEFC9-E010-4BE6-AAC5-B201BED399A7  | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 14               | B6FEFC9-E010-4BE6-AAC5-B201BED399A7  | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |
| 15               | 0C85941E-2AA6-4FC2-A757-C9CB1BCE0F   | Extract Tickets | 0050DCA8-27C5-4368-AB3F-D37A58D82473 | 1       | 1         | 0         |

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, a database named 'jobaccount' is selected. In the center pane, a query results grid displays data from the 'jobs.jobs\_execution' table. The grid has columns: job\_name, job\_id, job\_version\_number, step\_id, is\_active, lifecycle, and create\_time. There are three rows of data, all with 'Lifecycle' set to 'Succeeded'. The rows correspond to three different job IDs: AC0A07E9. The bottom status bar indicates the query was executed successfully and took 0:00:00 to run, returning 3 rows.

|   | 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 |

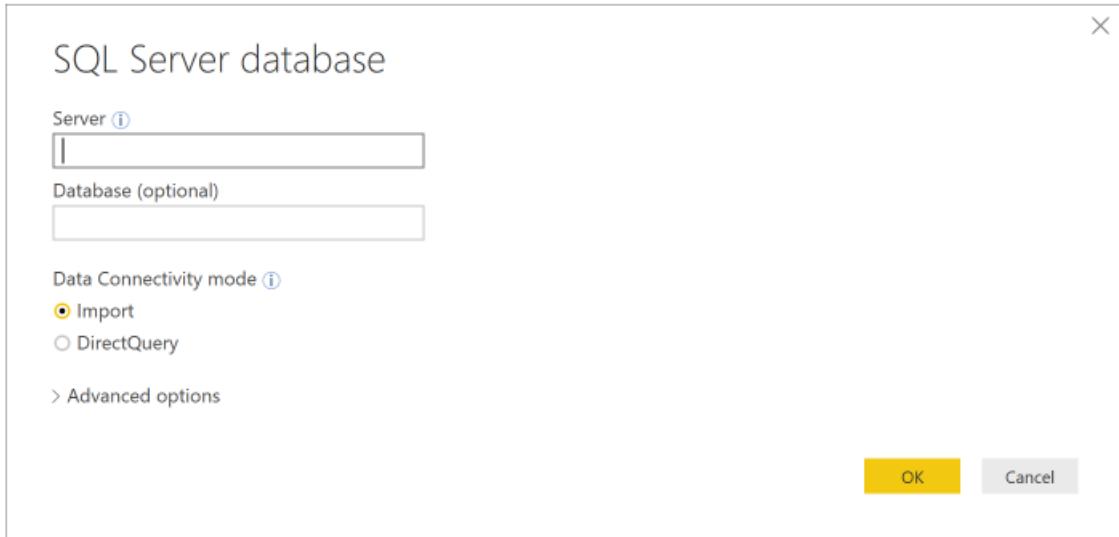
## Data exploration

### Visualize tenant data

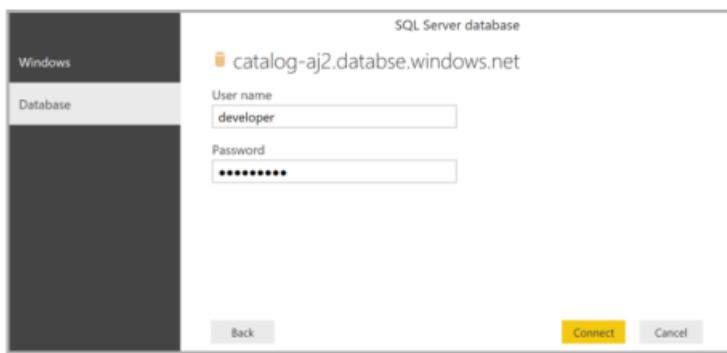
The data in the star-schema table provides all the ticket sales data needed for your analysis. To make it easier to see trends in large data sets, you need to visualize it graphically. In this section, you learn how to use **Power BI** to manipulate and visualize the tenant data you have extracted and 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.



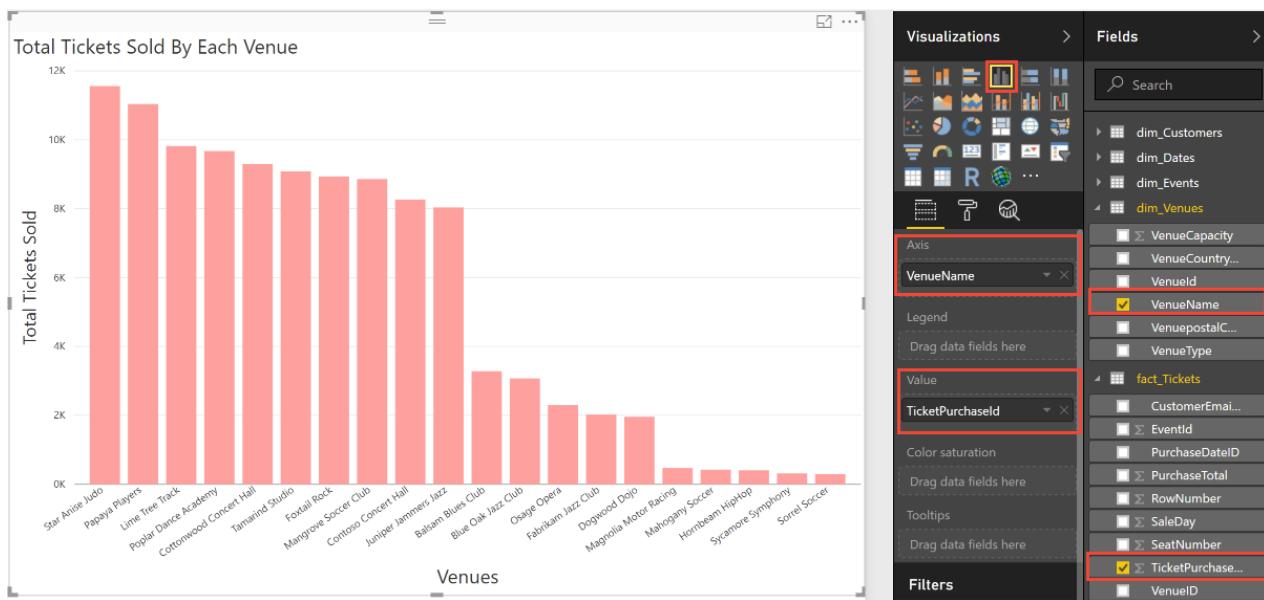
5. Select **Database** in the left pane, then enter user name = *developer*, and enter password = *P@ssword1*. Click **Connect**.



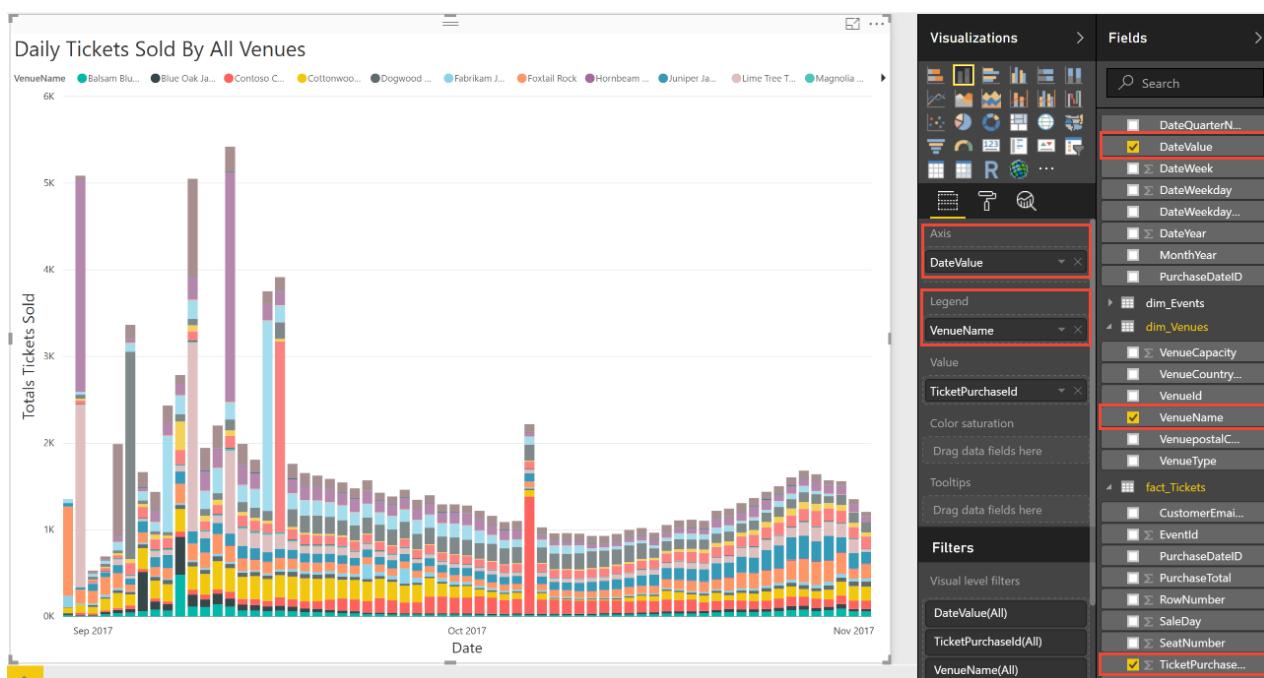
6. In the **Navigator** pane, under the analytics database, select the star-schema tables: fact\_Tickets, dim\_Events, dim\_Venues, dim\_Customers and dim\_Dates. Then select **Load**.

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.

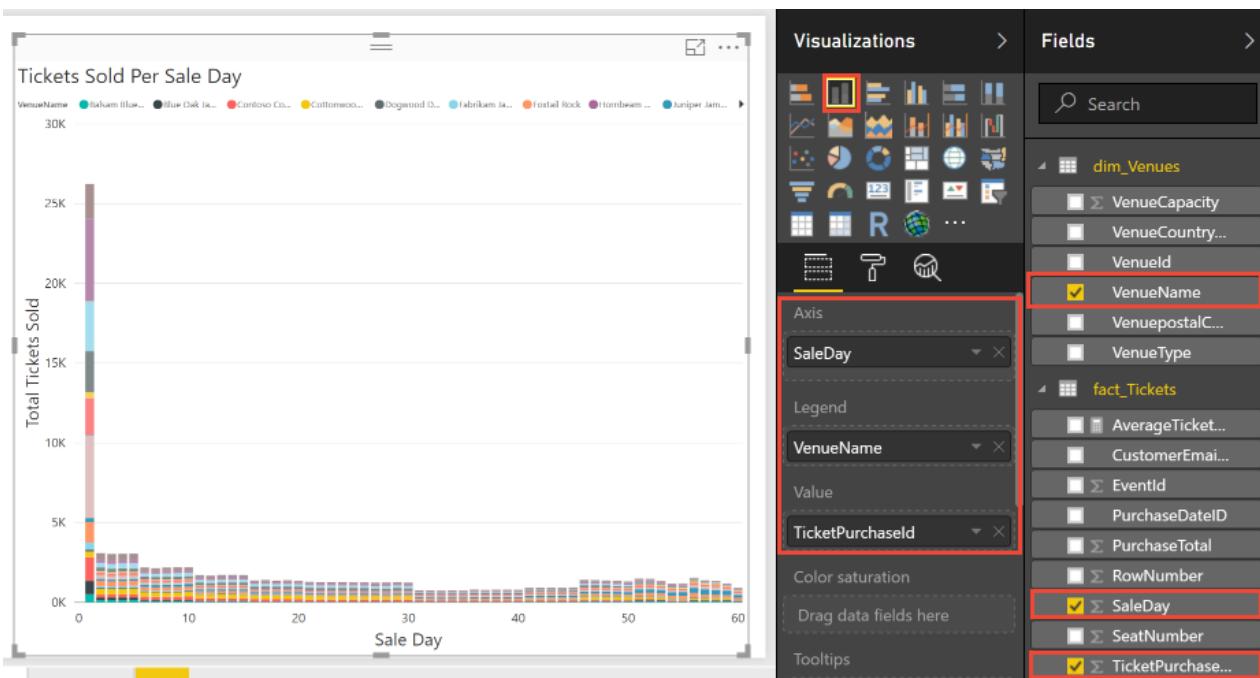
You start by analyzing ticket sales data to see the variation in usage across the venues. Select the following options in Power BI to plot a bar chart of the total number of tickets sold by each venue. Due to random variation in the ticket generator, your results may be different.



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.

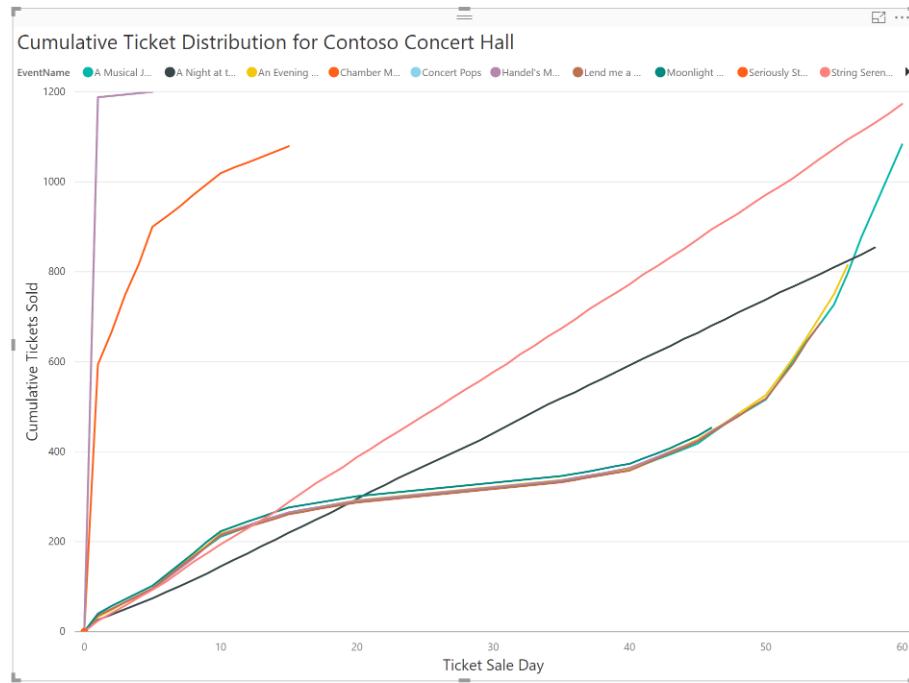


Next you want to further investigate the significance of these peak sale days. When do these peaks occur after tickets go on sale? To plot tickets sold per day, select the following options in Power BI.



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.

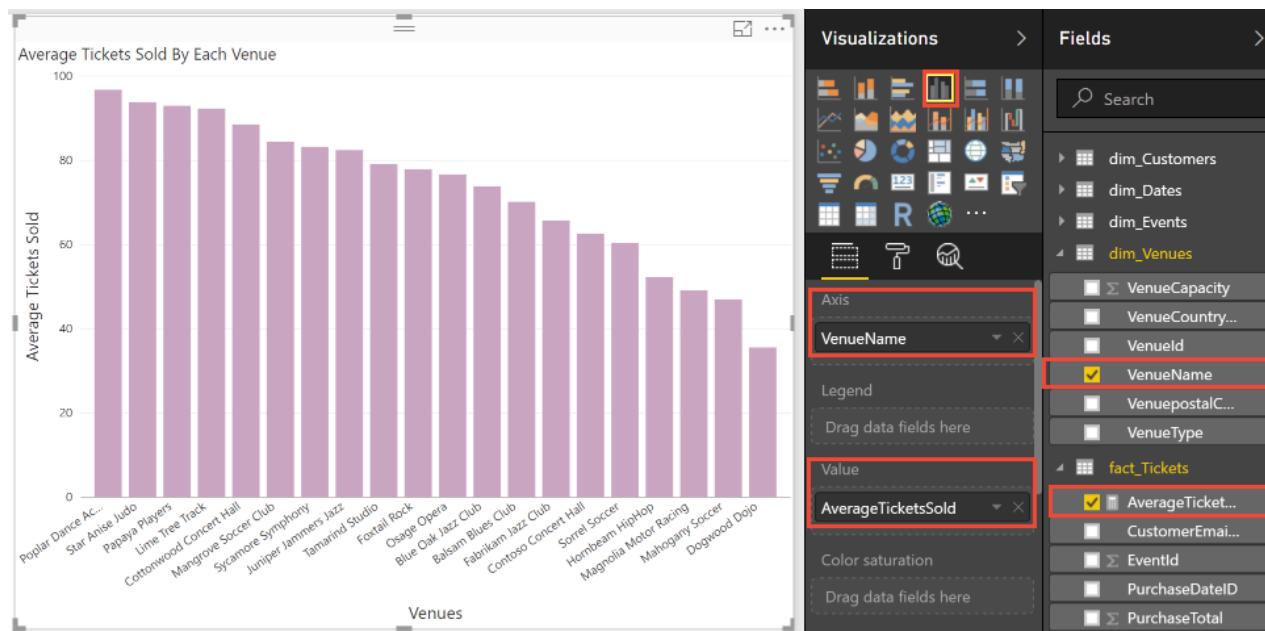
The insights into ticket selling patterns might lead Wingtip Tickets to optimize their business model. Instead of charging all tenants equally, perhaps Wingtip should introduce service tiers with different compute sizes. Larger venues that need to sell more tickets per day could be offered a higher tier 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 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

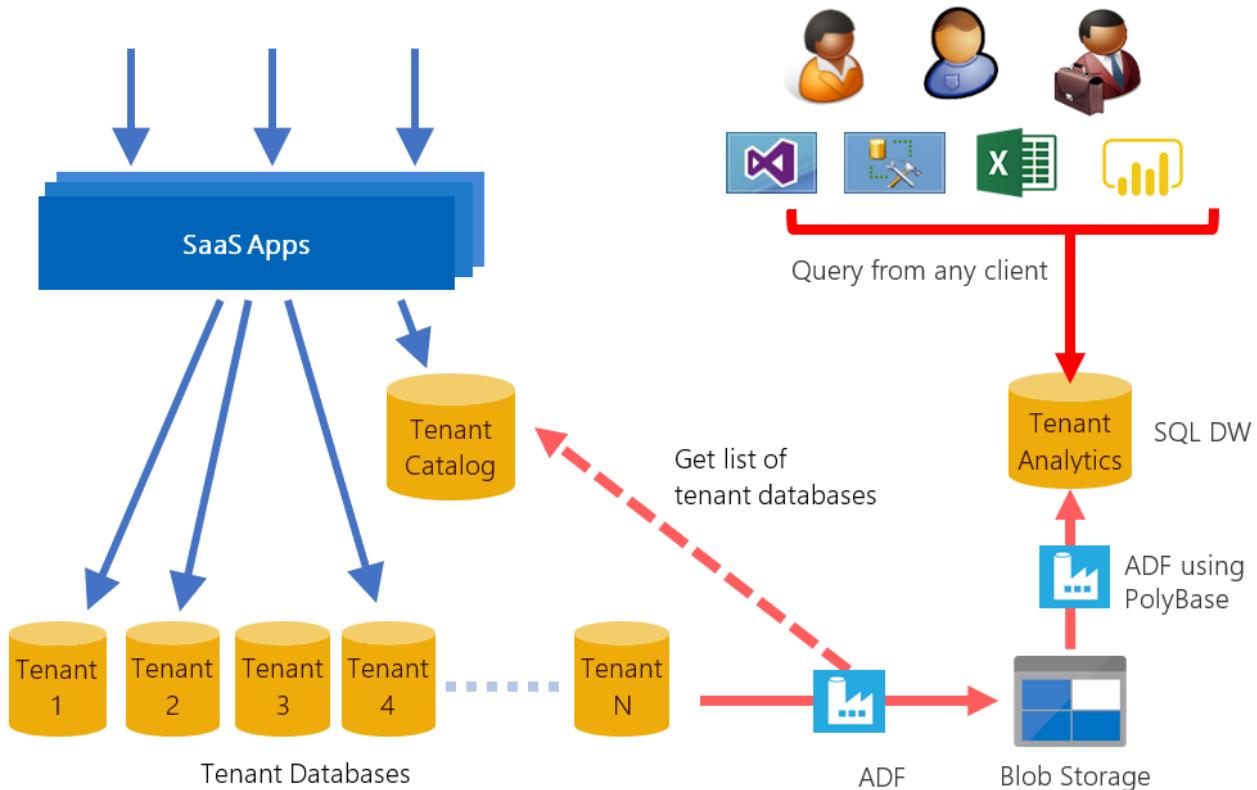
9/24/2018 • 16 minutes to read • [Edit Online](#)

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.

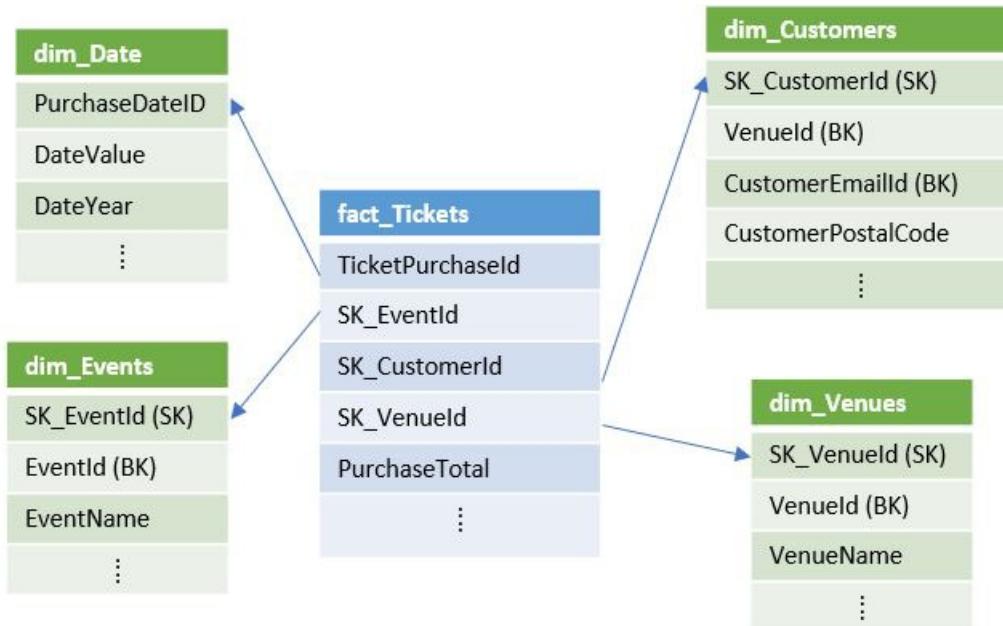
Accessing the data for all tenants is simple when all the data is in just one multi-tenant database. But access is more complex when distributed at scale across thousands of databases. One way to tame the complexity is to extract the data to an analytics database or a data warehouse for query.

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.

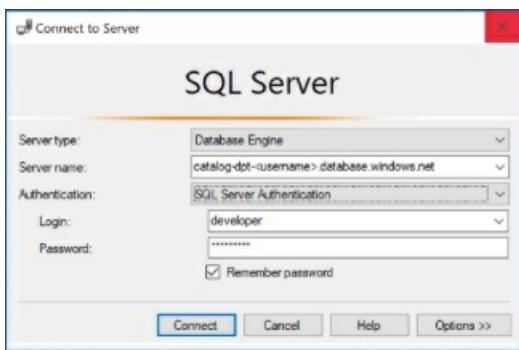
In this step, you deploy the additional resources used in the tutorial: a SQL Data Warehouse called *tenantanalytics*, an Azure Data Factory called *dbtodwload-<user>*, and an Azure storage account called *wingtipstaging-<user>*. The storage account is used to temporarily hold extracted data files as 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:

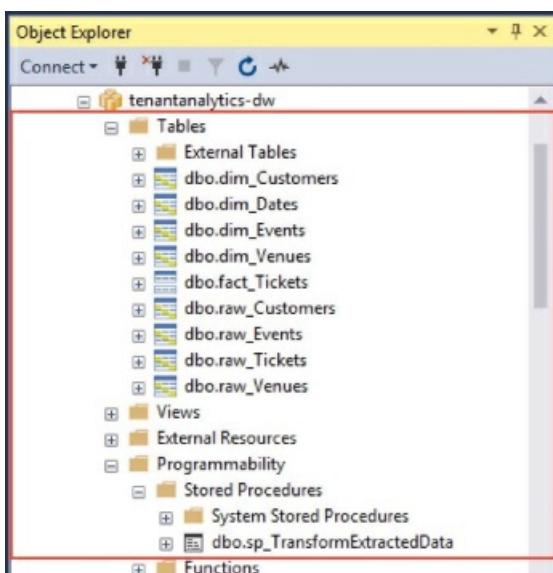
### Tenant databases and analytics store

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"/> | sycamoresymphony ( <i>tenants1-dpt-bgd/sycamoresymphony</i> ) | SQL database    | East US 2 |
|--------------------------|---|-----------------|-----------|
| <input type="checkbox"/> | tamarindstudio ( <i>tenants1-dpt-bgd/tamarindstudio</i> )     | SQL database    | East US 2 |
| <input type="checkbox"/> | wingtipstagingbgd   | 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.

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.

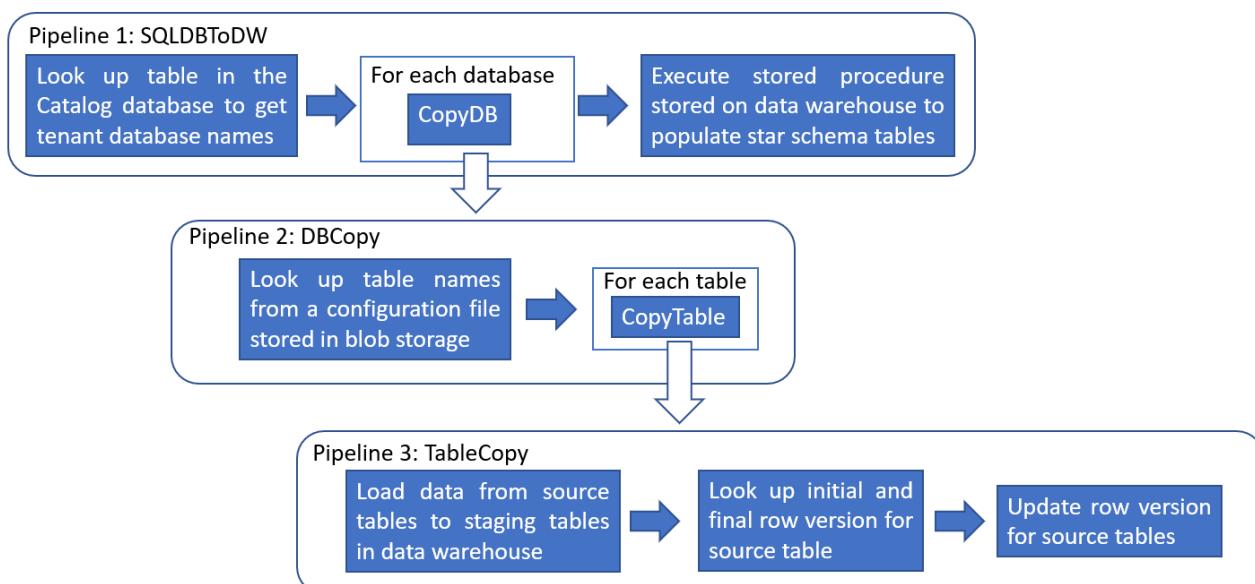
## Extract, Load, and Transform data

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**.

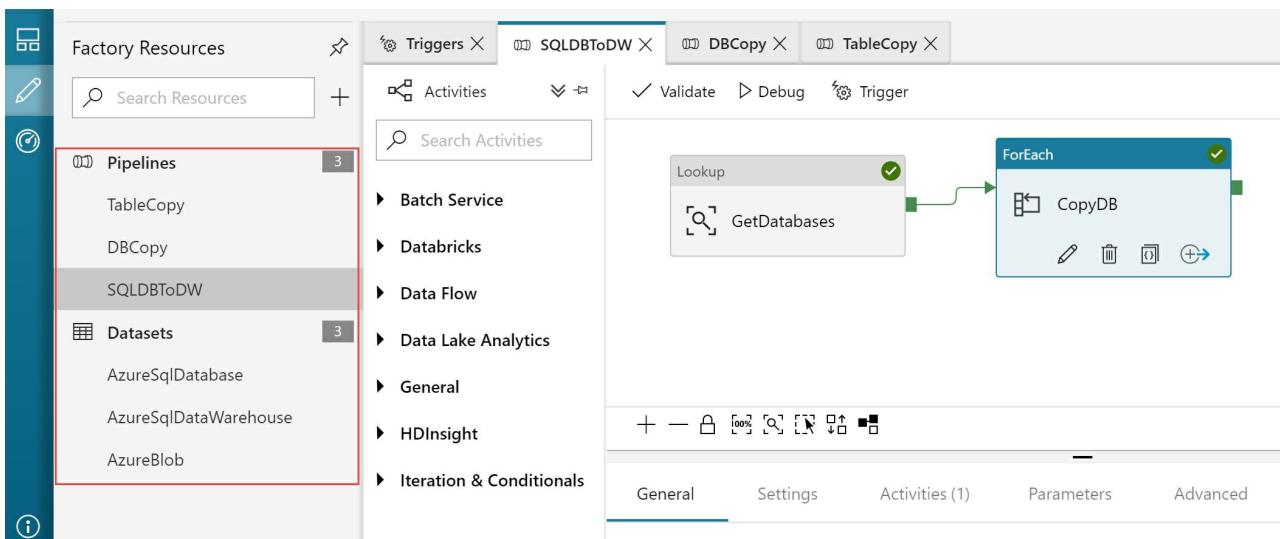
In the previous section, you deployed and initialized the necessary Azure resources, including the data factory. The deployed data factory includes the pipelines, datasets, linked services, etc., required to extract, load, and transform the tenant data. Let's explore these objects further and then trigger 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:

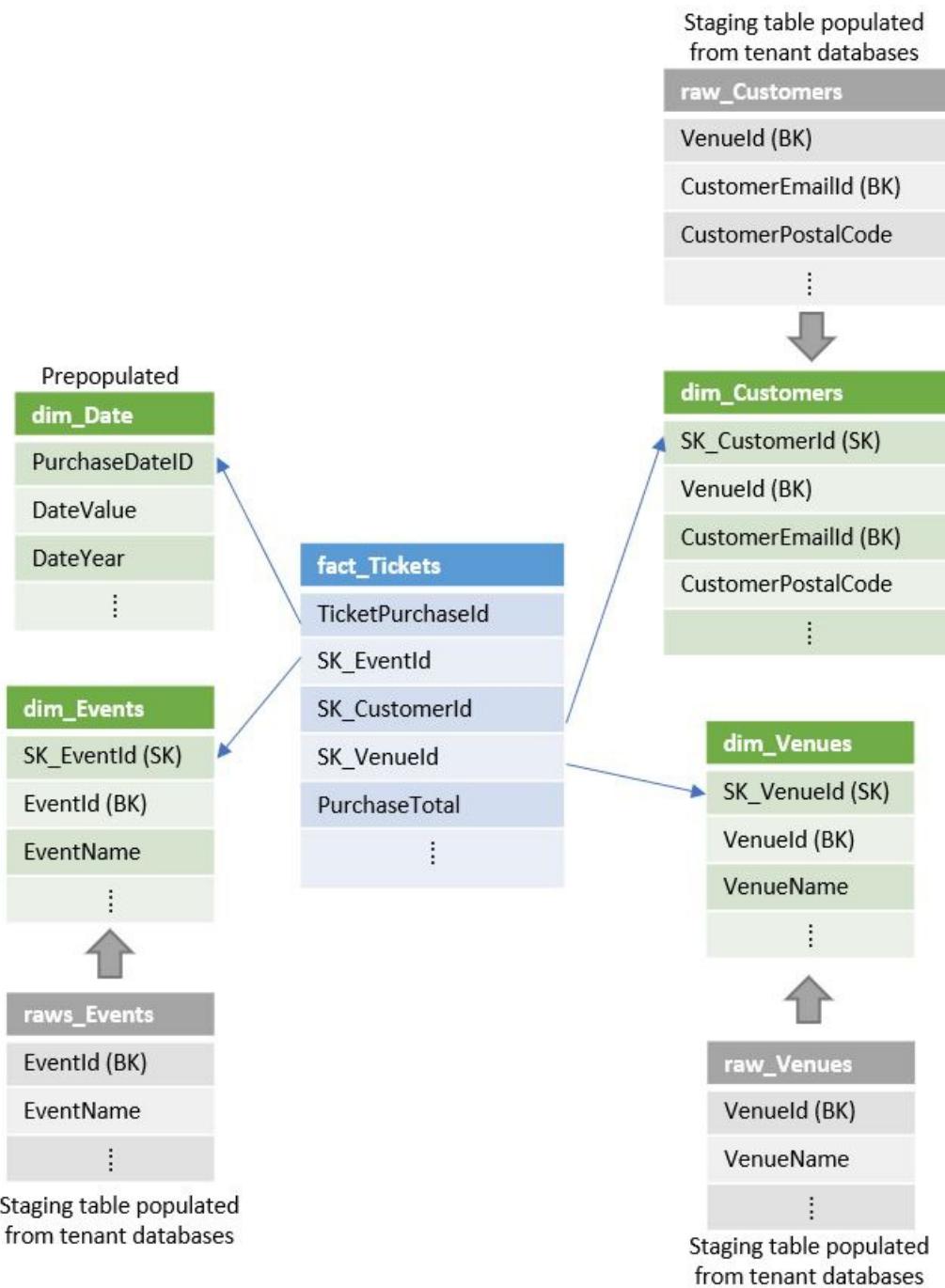
The screenshot shows the 'Connections' blade in the Azure Data Factory interface. On the left, there's a sidebar with icons for Home, Create, Pipelines, Datasets, and Connections. The main area has tabs for 'Linked Services' and 'Integration Runtimes', with 'Linked Services' selected. A search bar at the top says 'Search Resources'. Below it, a '+ New' button is followed by a table listing three connections:

| 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.

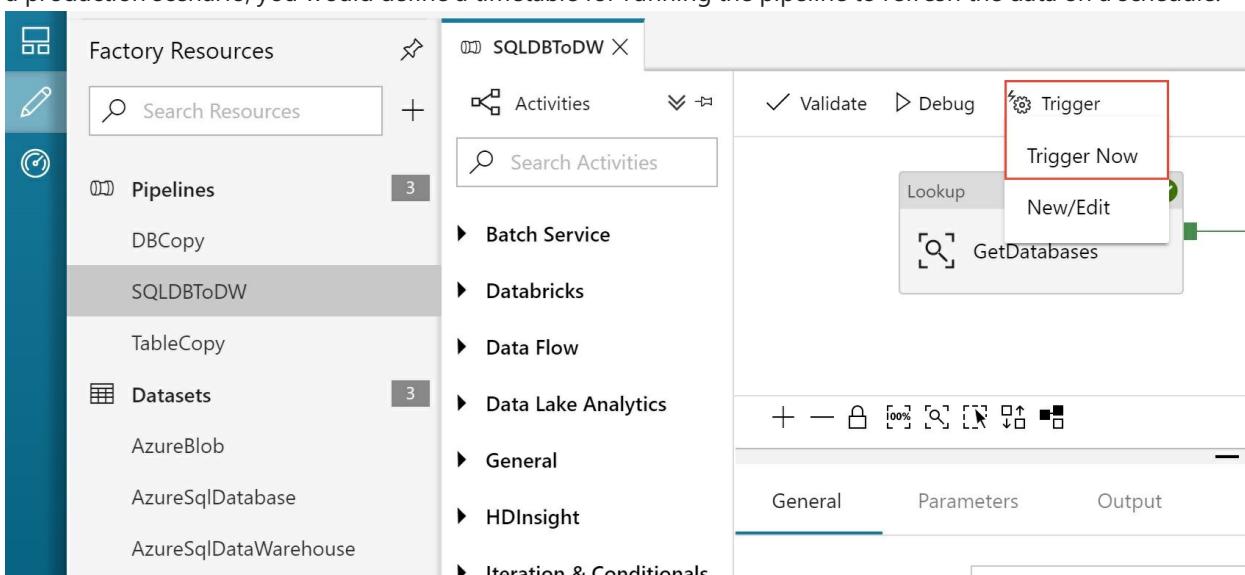
The final step of the transform deletes the staging data ready for the next execution of the pipeline.

### 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

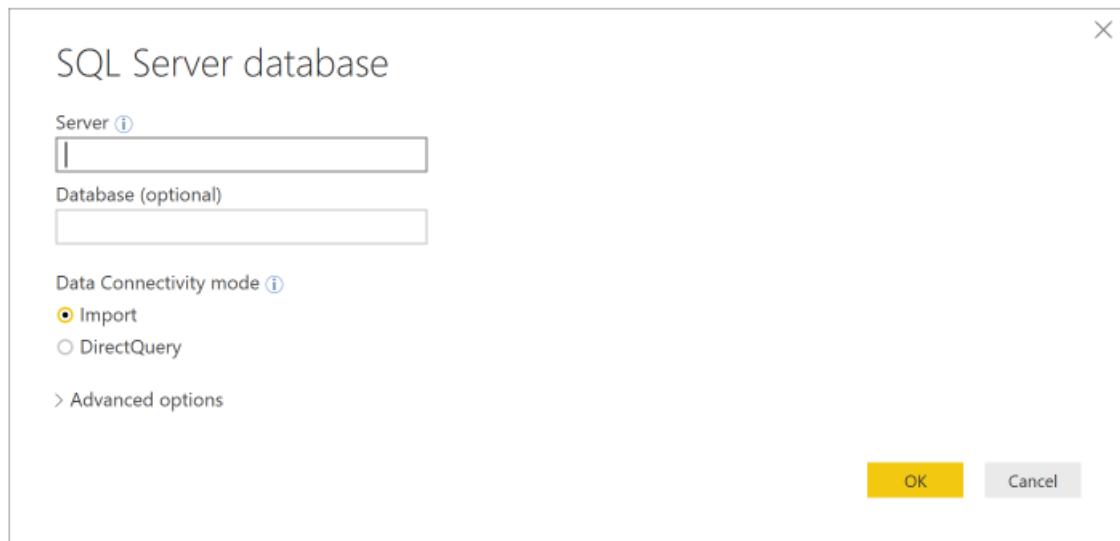
### Visualize tenant data

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**.



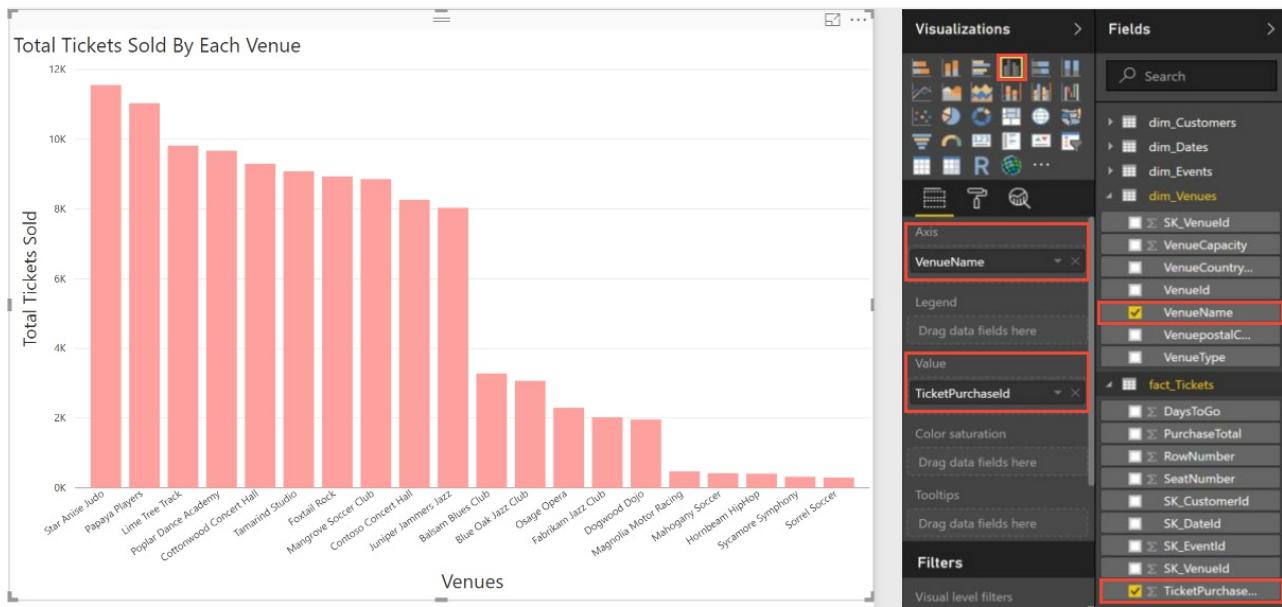
5. Select **Database** in the left pane, then enter user name = *developer*, and enter password = *P@ssword1*. Click **Connect**.



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**.

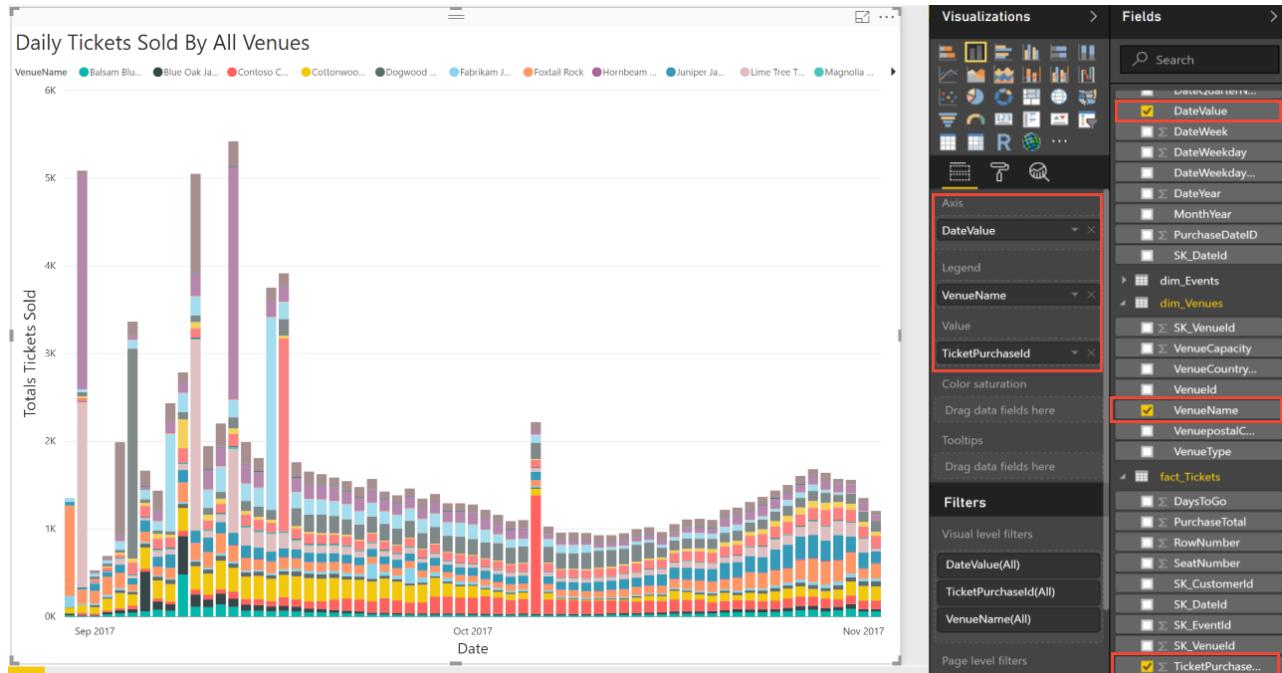
Congratulations! You successfully loaded the data into Power BI. Now explore interesting visualizations to gain insights into your tenants. Let's walk through how analytics can provide some data-driven recommendations to the Wingtip Tickets business team. The recommendations can help to optimize the 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.)



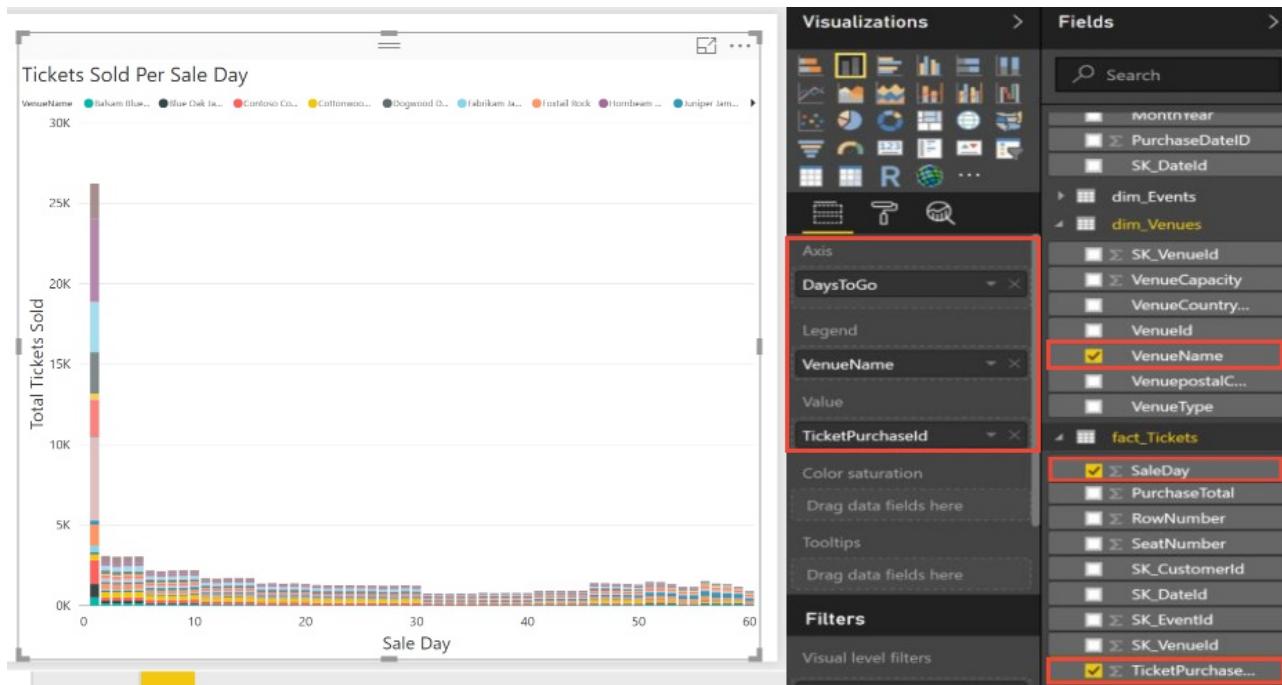
The preceding plot confirms that the number of tickets sold by each venue varies. Venues that sell more tickets are using your service more heavily than venues that sell fewer tickets. There may be an opportunity here to tailor resource allocation according to different tenant needs.

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.



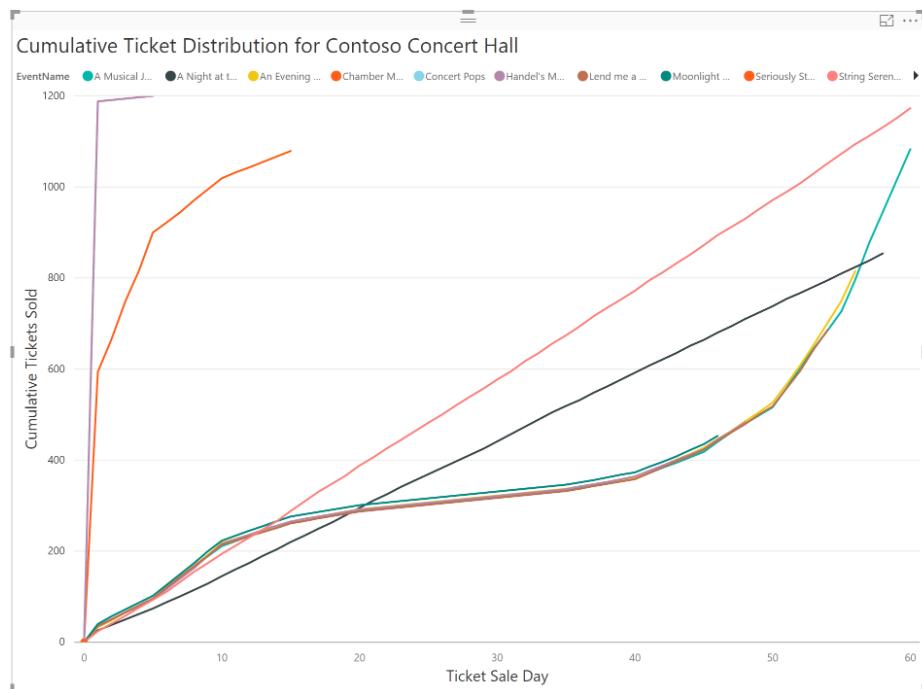
The preceding chart shows that ticket sales spike for some venues. These spikes reinforce the idea that some venues might be consuming system resources disproportionately. So far there is no obvious pattern in when the spikes occur.

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.

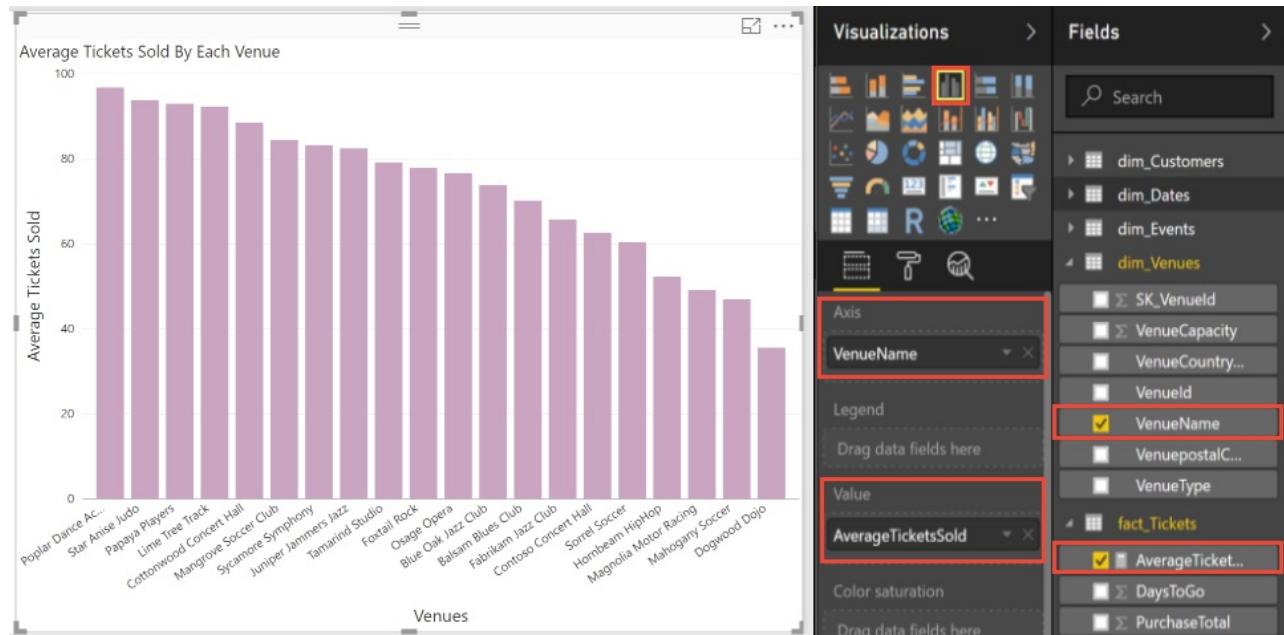
The insights into ticket selling patterns might lead Wingtip Tickets to optimize their business model. Instead of charging all tenants equally, perhaps Wingtip should introduce service tiers with different compute sizes. Larger venues that need to sell more tickets per day could be offered a higher tier 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 **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 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!

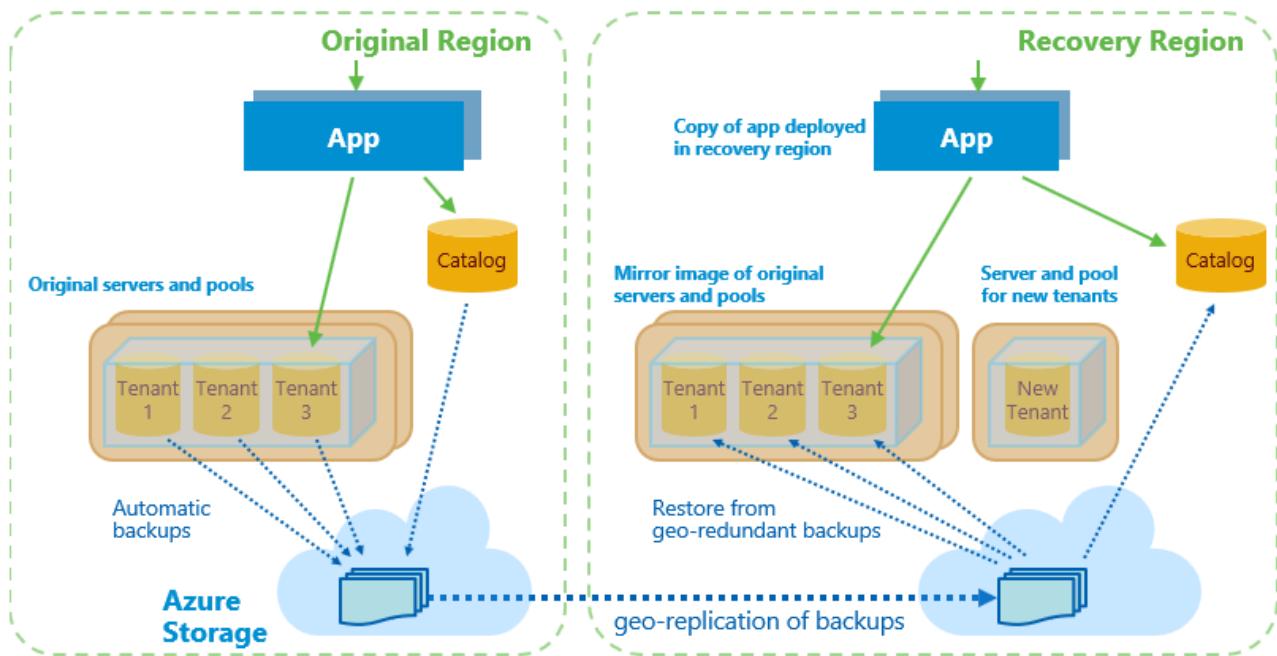
## Additional resources

- Additional [tutorials that build upon the Wingtip SaaS application](#).

# Use geo-restore to recover a multitenant SaaS application from database backups

10/16/2018 • 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". At the bottom of the page, a red box highlights the footer text: "Tenant id: 1976168774 | raw: 0xF5C9F146", "Database: contosoconehmerhall", "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

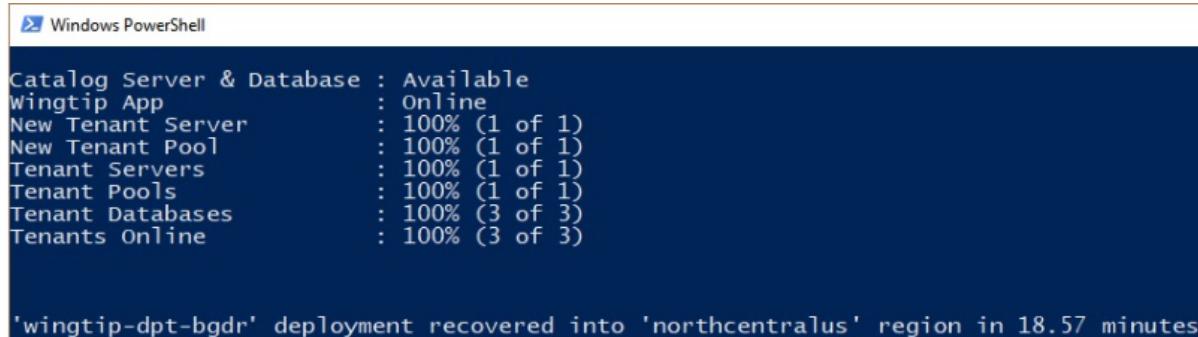
This tutorial restores databases from geo-redundant backups. Although these backups are typically available within 10 minutes, it can take up to an hour. The script pauses until they're available.

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><b>MAR 20 TUE</b></p> <p>Event 3<br/>Performer 3</p> | <p><b>MAR 23 FRI</b></p> <p>Event 4<br/>Performer 4</p> | <p><b>Tickets</b></p> <p><b>mytickets</b><br/>Update your list of favorites and never miss an event!</p> <p><b>Sign In</b></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.

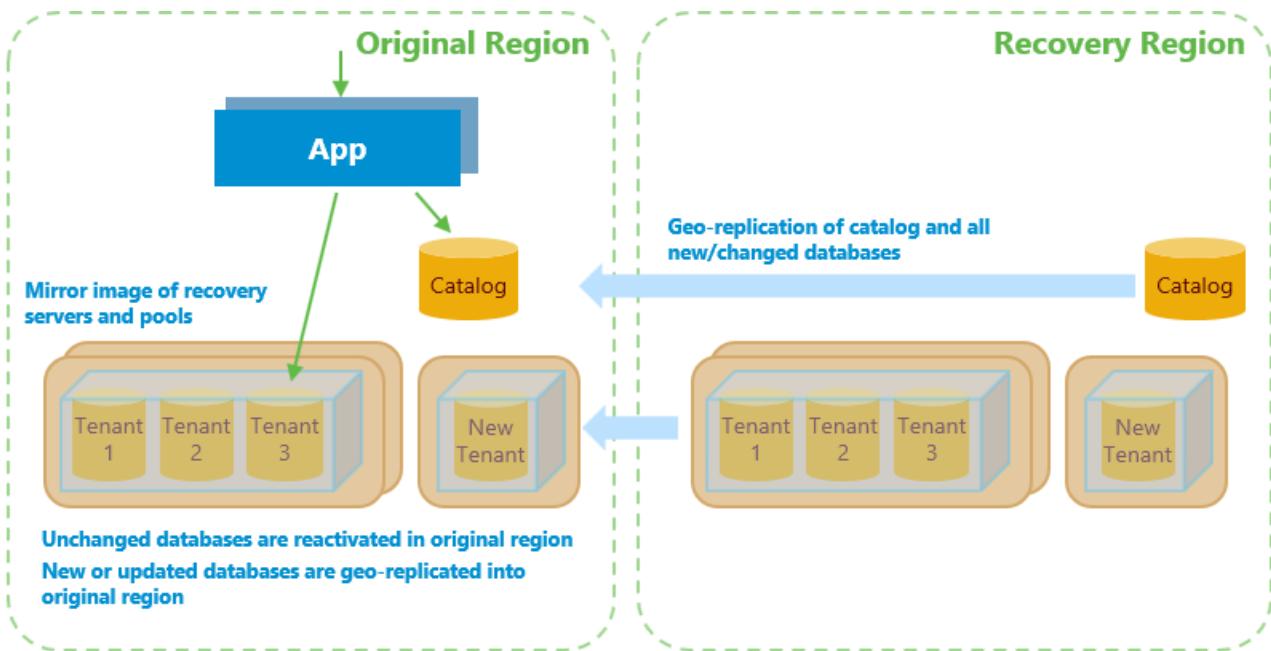
At this point in the tutorial, you have recovered the application, which is now running in the recovery region. You have provisioned a new tenant in the recovery region and modified data of one of the restored tenants.

### 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

9/24/2018 • 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.

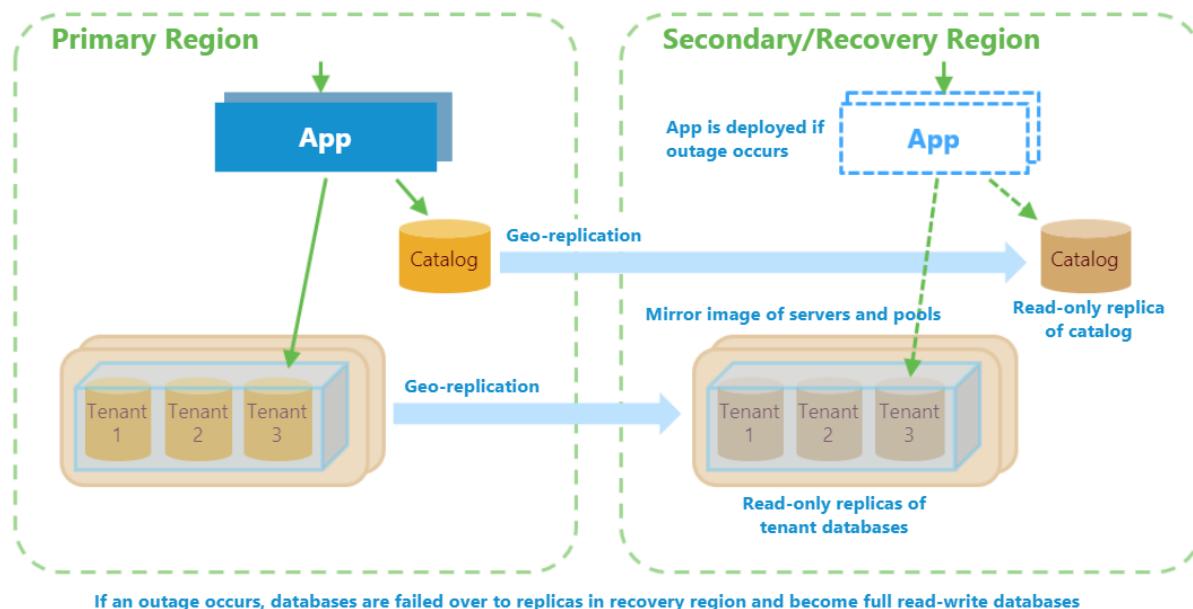
This tutorial explores both the failover and fallback workflows. You'll learn how to:

- 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 single 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 single 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 bar with the Wingtip Tickets logo and the text "Events Hub". Below the header is a search bar and a dropdown menu labeled "Venues" containing "Contoso Concert Hall", "Dogwood Dojo", and "Fabrikam Jazz Club". A small orange arrow icon is next to the dropdown menu. At the bottom of the page, there's a footer bar with the Wingtip Tickets logo, the text "Running on Azure SQL Database. © 2018 Microsoft", and a red-bordered box containing "Catalog database: tenantcatalog", "Server: catalog-dpt-bgdr2", and "Location: northcentralus". To the right of the footer, there's a link "Learn how to build a SaaS app on SQL database".

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.

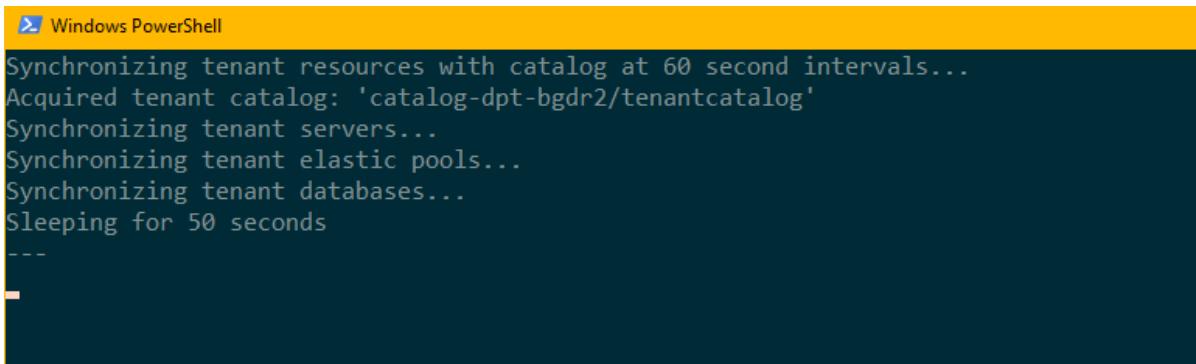
## Sync tenant configuration into catalog

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.

Select a region on the map or from the Target Regions list to create a secondary database.

**PRIMARY**

|                                     |                  |                                      |
|-------------------------------------|------------------|--------------------------------------|
| <input checked="" type="checkbox"/> | South Central US | tenants1-dpt-bgdr/contosoconcerthall |
|-------------------------------------|------------------|--------------------------------------|

**SECONDARIES**

|                                     |                  |   |
|-------------------------------------|------------------|---|
| <input checked="" type="checkbox"/> | North Central US | tenants1-dpt-bgdr-recovery/contosoconcerthall |
|-------------------------------------|------------------|---|

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 an orange 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". There's a search bar labeled "search". A list titled "Venues" shows 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".

The screenshot shows the Wingtip Tickets Platform homepage. It has an orange header with a ticket icon and the text "Wingtip Tickets Platform". Below the header is a dark grey footer bar with the text "Running on Azure SQL Database. © 2018 Microsoft". To the right of the footer is a red-bordered box containing "Catalog database: tenantcatalog", "Server: catalog-dpt-bgdr2-recovery", and "Location: southcentralus". At the bottom right, there's a link "Learn how to build a SaaS app on SQL database".

- 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. It has an orange header with a ticket icon and the text "Contoso Concert Hall". The main content area features a large text "We'll be back soon!" and a message: "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." Below the message is 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 portal interface for the 'Wingtip-bgdr2-recovery' resource group. 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 content area displays the 'Essentials' blade for the resource group. It features a search bar, filter options (Filter by name..., All types, All locations, No group), and a table listing 13 items. The table columns are NAME, TYPE, and LOCATION. The listed items include:

| 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 sections including Pricing tier (scale DTUs), Geo-Replication (which is selected), Auditing & Threat Detection, Vulnerability Assessment, Data discovery & classification, Dynamic Data Masking, Transparent data encryption, Connection strings, and Sync to other databases. The main content area displays a world map with green circles representing available regions for geo-replication. Below the map, the "PRIMARY" section shows "North Central US" with the connection string "tenants1-dpt-bgdr-recovery/contosoconcerthall". The "SECONDARIES" section shows "South Central US" with the connection string "tenants1-dpt-bgdr/contosoconcerthall". A message 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

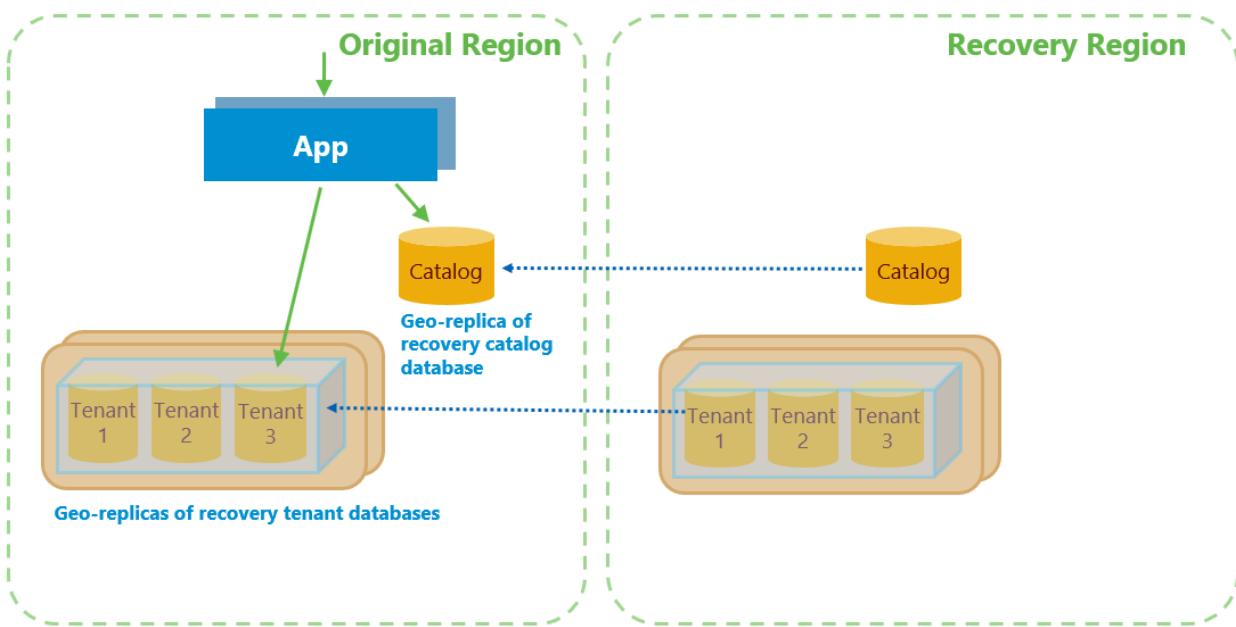
In this task, you update one of the tenant databases.

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



## 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

10/16/2018 • 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 (Her 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

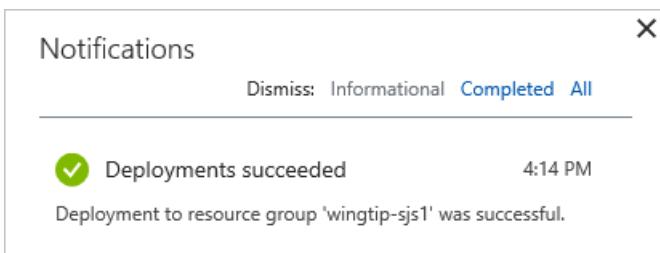
For this demonstration, do not use any pre-existing resource groups, servers, or pools. Instead, choose **Create a new resource group**. Delete this resource group when you are finished with the application to stop related billing. Do not use this application, or any resources it creates, for production. 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.

ALL SESSIONS

WELCOME [SIGN IN]

Next | Jazz Grabbs You

Tickets >

my|tickets

Update your list of favorites and never miss an event!

Sign In

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. Next to it is the text "Red Maple Racing". On the far right are links for "WELCOME [SIGN IN]" and "ALL RACES". The main image is a blurred photograph of a race car cockpit with a driver wearing a helmet. Below the image are buttons for "Next | Event 3" and "Tickets &gt;". To the left, there's a blue box with the date "OCT 21 SAT" and the text "Event 3" and "Performer 3". To the right, there's a box titled "my|tickets" with 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'. On the left, there's a navigation sidebar with various icons and links like Overview, Activity log, Access control (IAM), Tags, SETTINGS, Quickstart, Resource costs, Deployments, Policies, Properties, Locks, and Automation script. The main content area displays a list of resources under the 'Essentials' category. A search bar at the top allows filtering by name. The list includes:

| 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 |

2. 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 has 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, and Auditing & Threat Detection. The main content area shows the server's properties and a list of databases.

**Properties:**

- Resource group (change): 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

**SQL databases:**

| DATABASE      | STATUS | PRICING TIER |
|---------------|--------|--------------|
| basetenantdb  | Online | Standard: S0 |
| tenantcatalog | Online | Standard: S0 |

3. 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

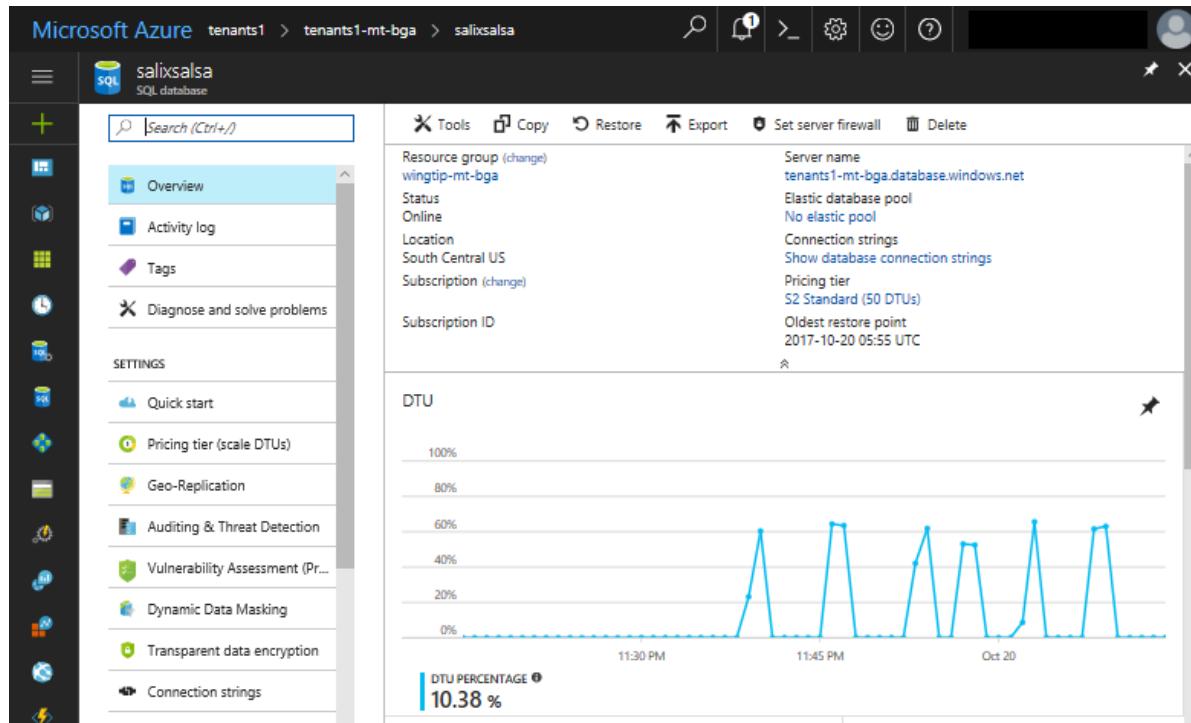
If the load generator has been running for several minutes, enough telemetry is available to look at the database monitoring capabilities built into the Azure portal.

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

9/24/2018 • 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.

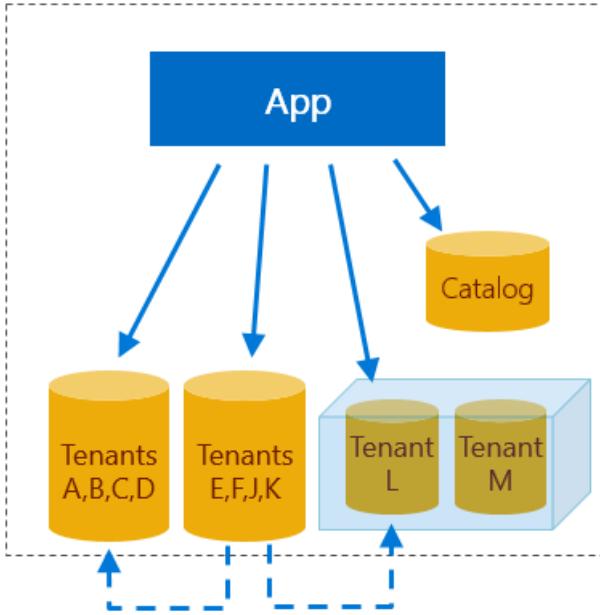
In this multi-tenant sharded model, the table schemas inside each database include a tenant key in the primary key of tables that store tenant data. The tenant key enables each individual database to store 0, 1, or many tenants. The use of sharded databases makes it easy for the application system to 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.

The screenshot shows the Windows PowerShell ISE interface. The title bar reads "Windows PowerShell ISE". The menu bar has "File", "Edit", "View", "Tools", "Debug", "Add-ons", and "Help". The "Debug" menu is open, showing the following options:

- Step Over F10
- Step Into** F11 (highlighted)
- 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
- Display Call Stack Ctrl+Shift+D

The code editor window contains a PowerShell script named "Demo-ProvisionTenants.ps1". The script provisions a new tenant. Lines 38 and 39 are highlighted in yellow, and line 38 contains the word "New". The code includes comments and several cmdlets like Write-Output, Start-Process, and New-AzureRmResourceGroup.

```

30 Initial
31
32 $wtpUser
33 $config
34
35 ### Provision a new tenant
36 if ($sc)
37 {
38     New-AzureRmResourceGroup -Name $tenantName -Location $location
39
40     Write-Output "Provisioning complete for tenant '$TenantName'"
41
42     # Open the events page for the new venue
43     Start-Process "http://events.wingtip-mt.($wtpUser.Name).trafficmanag

```

5. Trace the script's execution using the **Debug** menu options, **F10** and **F11**, to step over or into called functions.

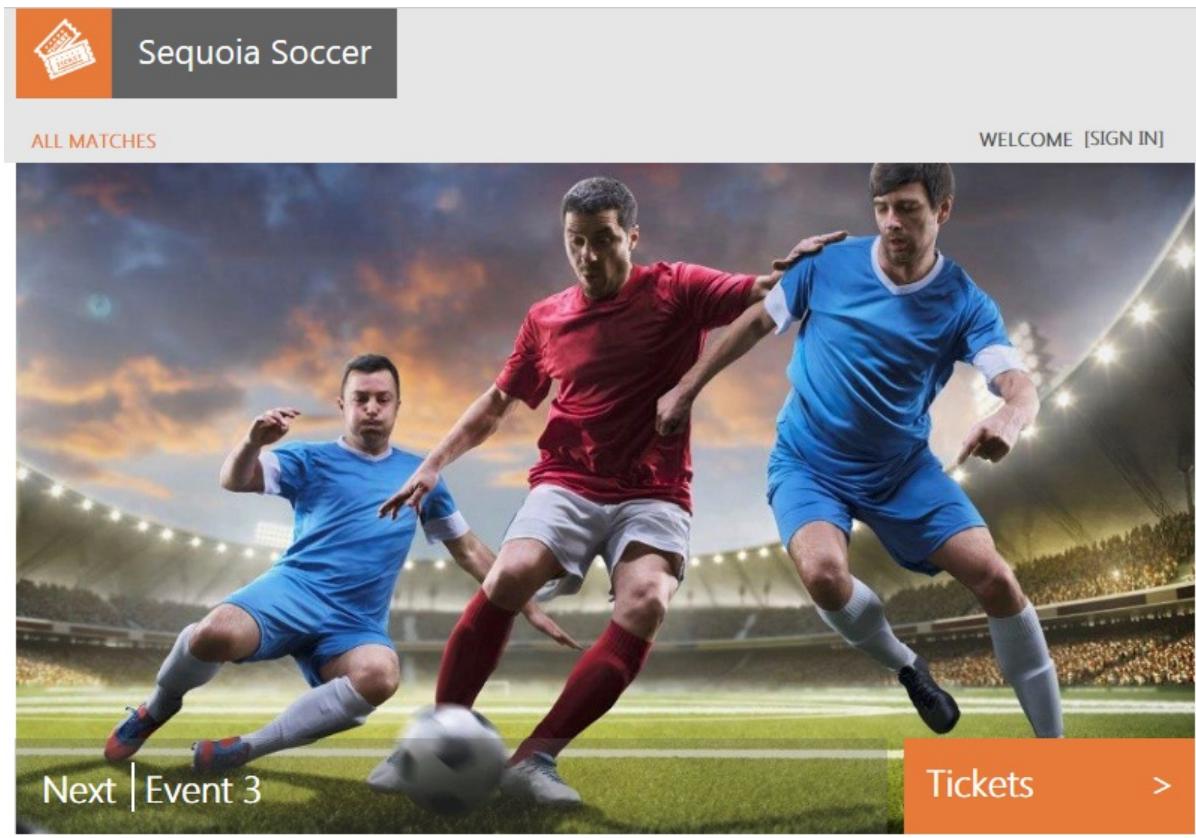
For more information about debugging PowerShell scripts, see [Tips on working with and debugging PowerShell scripts](#).

## 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.



#### 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 resource group (wingtip-mt-bgb), status (Available), location (South Central US), subscription (Standard), and a link to view recommended elastic database pool. 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

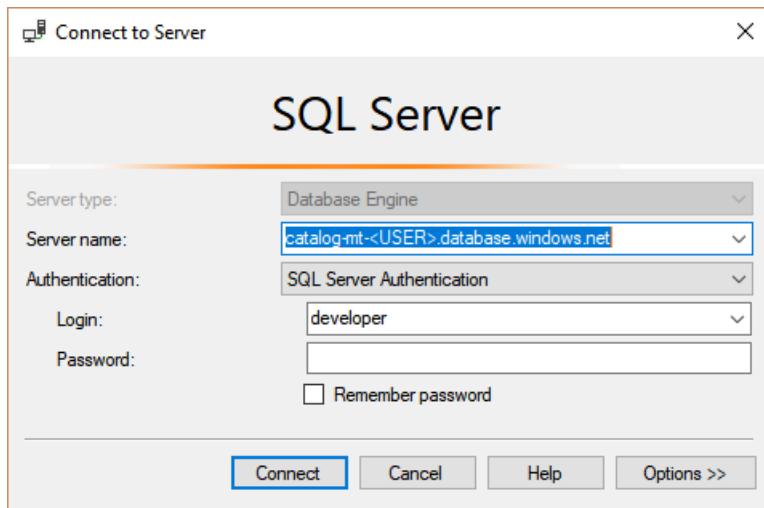
Open the Events Hub page in the browser (<http://events.wingtip-mt.<USER>.trafficmanager.net>)

#### 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

10/5/2018 • 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, [Log Analytics](#) can be used. This is a separate Azure service that provides analytics over emitted diagnostic logs and telemetry gathered in a log analytics workspace. Log Analytics 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 (approx 30 DTU)   |
| 3    | Generate load with longer bursts per tenant  |
| 4    | Generate load with higher DTU bursts per tenant (approx 70 DTU)  |
| 5    | Generate a high intensity (approx 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.

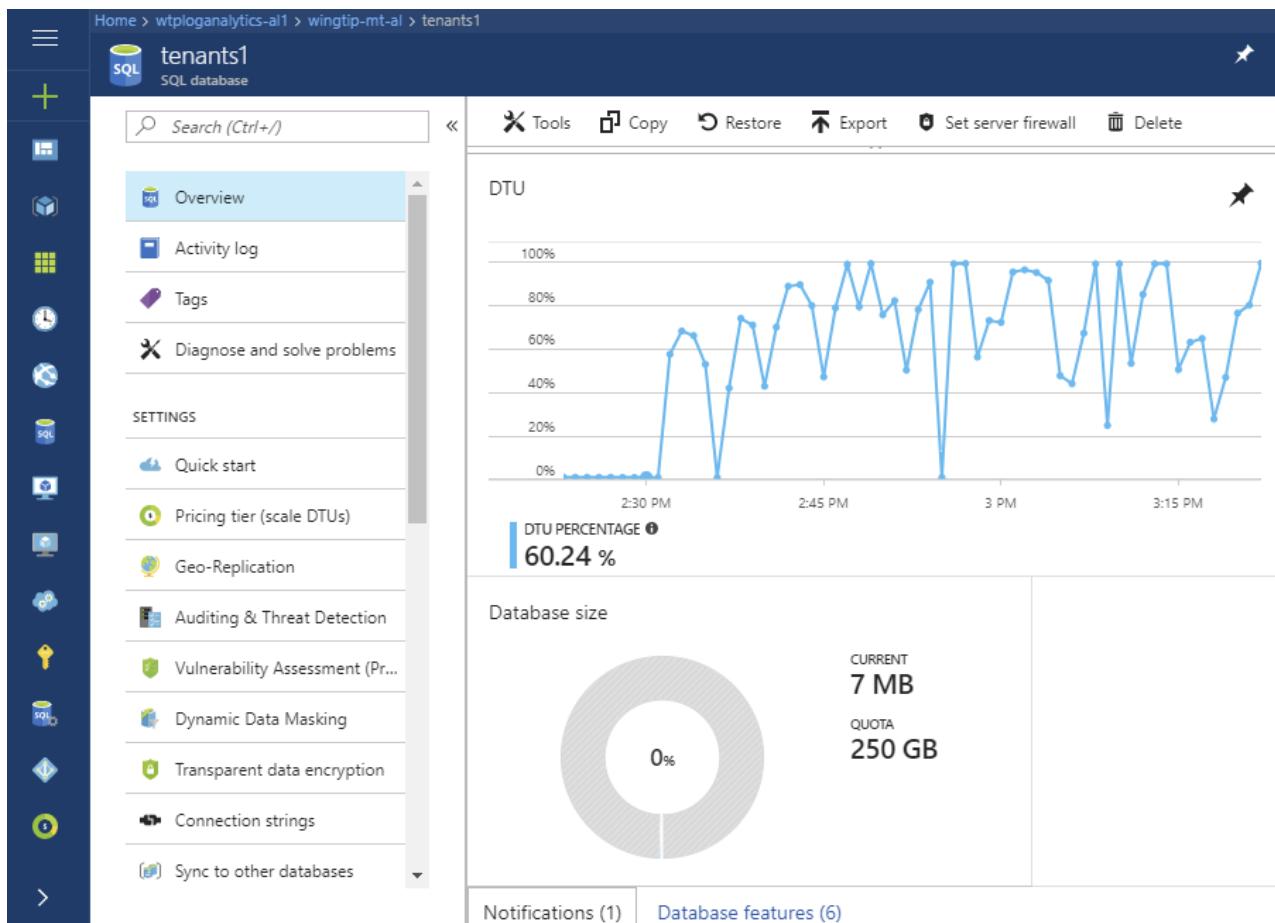
#### **IMPORTANT**

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

Search (Ctrl+ /) << **Add alert**

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**

**NAME**  
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, and Monitoring. A red box highlights the 'Alert rules' link under the Monitoring section. At the top right, there's a prominent 'Add alert' button with a plus sign, also enclosed in a red box. The overall theme is dark blue, typical of the Azure UI.

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 'Resource' dropdown is set to 'tenants1-ml-al2/tenants1 (servers/dat...)'. The 'Name' field is 'High DTU'. The 'Metric' dropdown is 'DTU percentage'. A line chart displays a sharp increase from approximately 0% to 85% DTU usage between 6 PM and 12 AM on October 26. The 'Condition' is 'greater than', the 'Threshold' is 75%, and the 'Period' is 'Over the last 30 minutes'. An 'OK' button is at the bottom.

## 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 a single 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 (approx 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

10/30/2018 • 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.

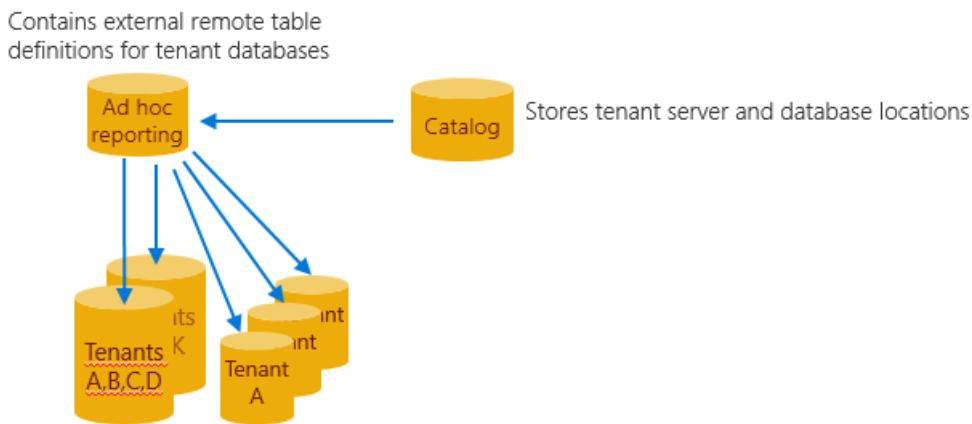
In this tutorial you learn:

- 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

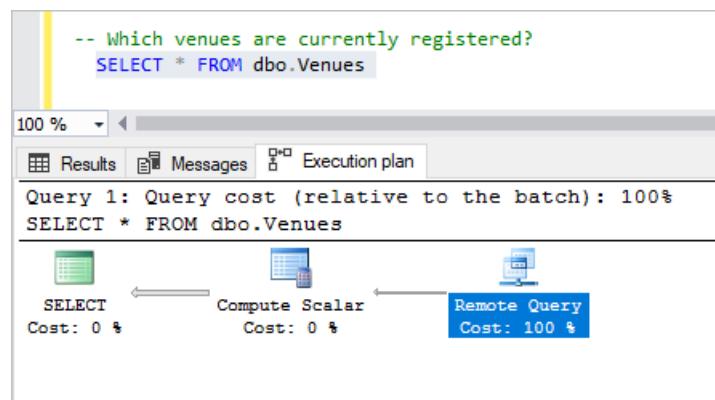
Now that the *adhocreporting* database is set up, go ahead and run some distributed queries. Include the execution plan for a better understanding of where the query processing is happening.

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,
       EventTypeName
  FROM dbo.Venues
     INNER JOIN dbo.VenueTypes ON Venues.VenueType = VenueTypes.VenueType
```

Execution plan:

```
00 % ▾
Results Messages Execution plan
Query 1: Query cost (relative to the batch): 100%
SELECT VenueName, VenueTypeName, EventTypeName FROM dbo.Venues INNER JOIN c
[...]

```

- 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
```

Execution plan:

```
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
[...]

```

## 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

9/24/2018 • 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

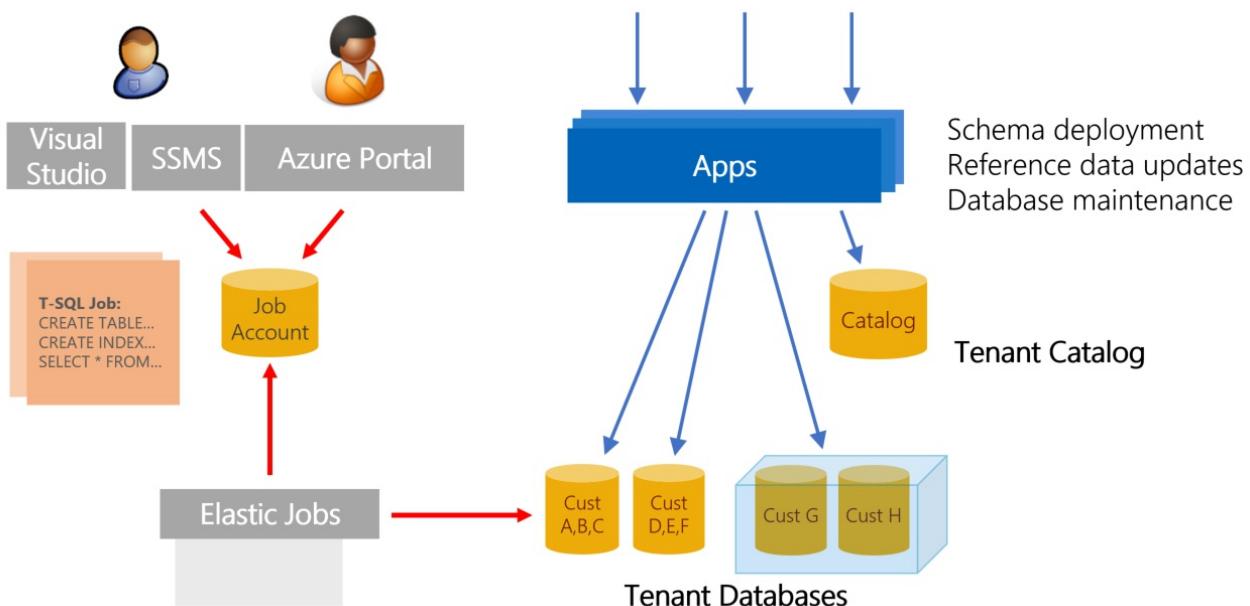
- The Wingtip Tickets multi-tenant database app must already be deployed:
  - For instructions, see the first tutorial, which introduces the Wingtip Tickets SaaS multi-tenant database app:  
[Deploy and explore a sharded multi-tenant application that uses Azure SQL Database](#).
    - The deploy process 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](#)
- [Create and manage 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

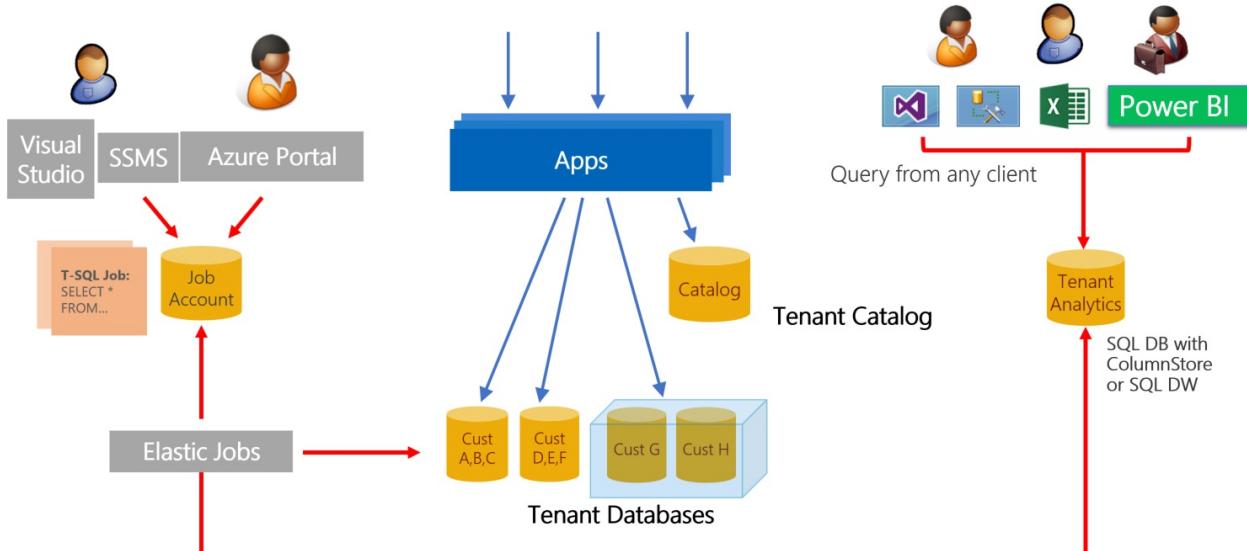
9/24/2018 • 13 minutes to read • [Edit Online](#)

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.

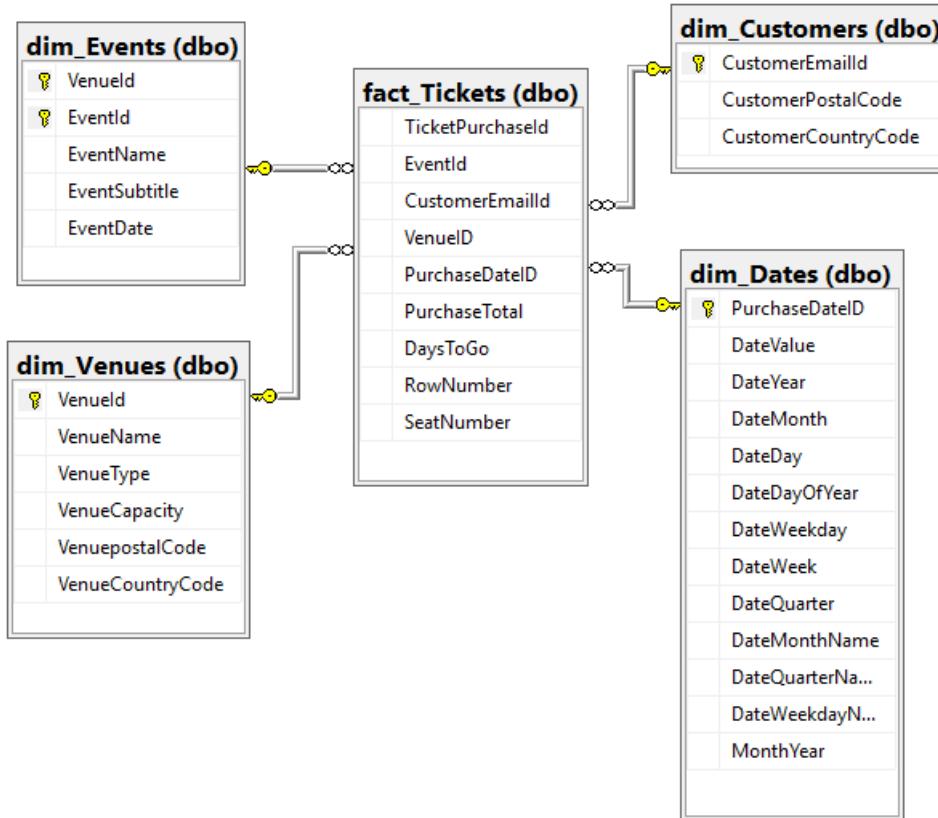
Accessing the data for all tenants is simple when all the data is in just one multi-tenant database. But the access is more complex when distributed at scale across thousands of databases. One way to tame the complexity is to extract the data to an analytics database or a data warehouse. You then query 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.

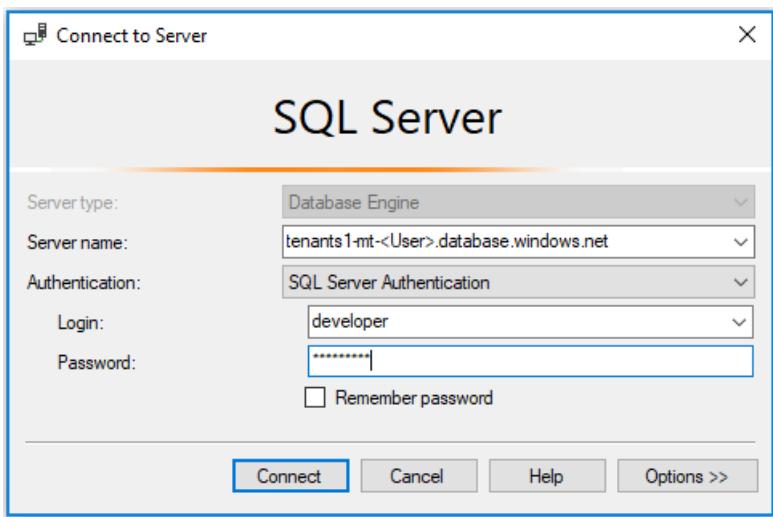
## Deploy the analytics store

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.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*.

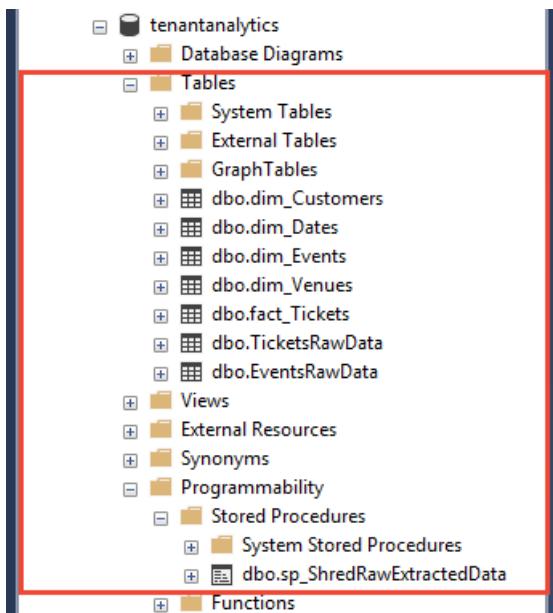


In the Object Explorer, perform the following steps:

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 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 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 the tenantanalytics database which contains the dbo.TicketsRawData table. The central pane displays a query results window for the ExtractTickets.sql script. The results show three rows of data in a table with columns: job\_execution\_id, job\_name, job\_id, job\_ver..., step\_id, is\_active, lifecycle, and create\_time. The data is as follows:

|   | 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  |

At the bottom of the results window, a message indicates: "Query executed successfully | catalog-mt-aj3.database.windows.net | developer (90) | jobaccount | 00:00:00 | 3 rows".

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 for 'catalog-mt-aj3.database.windows.net.jobaccount'. Several tables under the 'dbo' schema are highlighted with a red box: 'dbo.dim\_Customers', 'dbo.dim\_Dates', 'dbo.dim\_Events', 'dbo.dim\_Venues', 'dbo.EventsRawData', and 'dbo.fact\_Tickets'. The 'dbo.TicketsRawData' table is also visible. The central pane displays a T-SQL script named 'ShredRawExtractedD...t (developer (126))' which defines a job to shred raw data. The 'Results' tab at the bottom shows the execution status of the job, with three rows of data in the table:

| 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 |

The status bar at the bottom indicates 'Query executed successfully'.

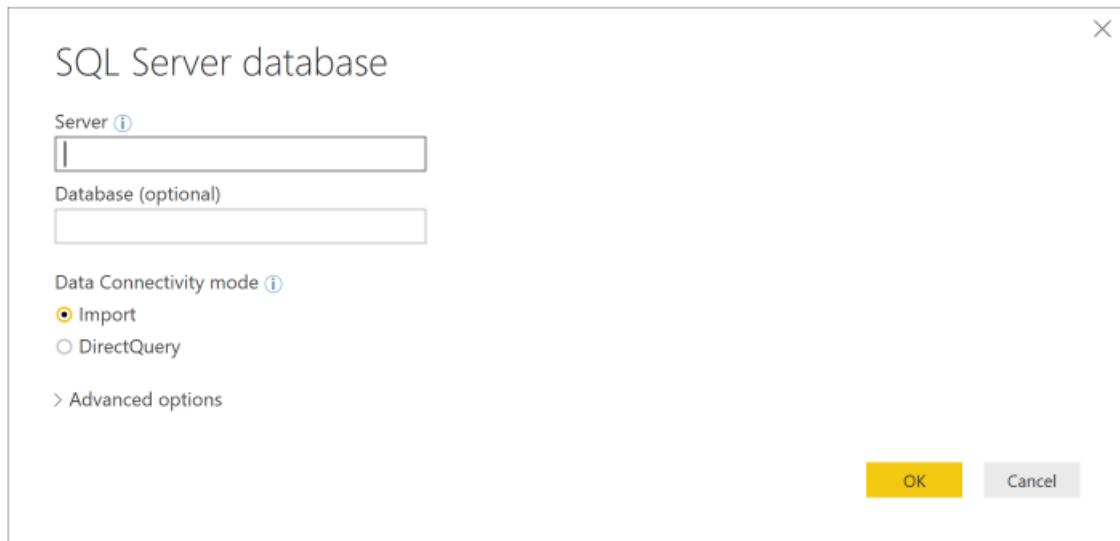
## Data exploration

### Visualize tenant data

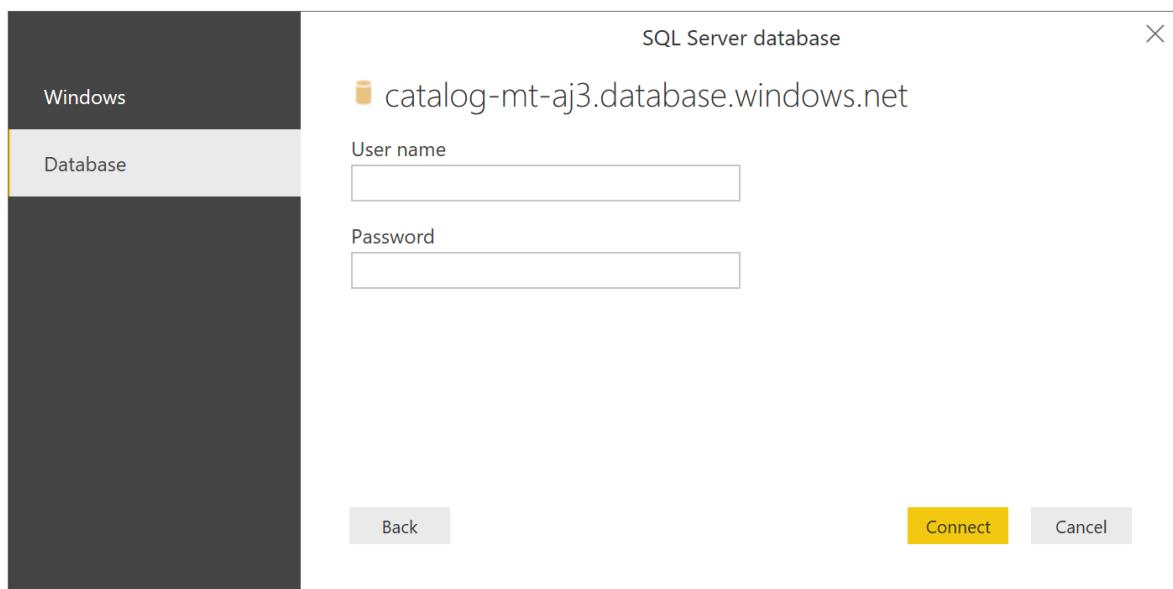
The data in the star-schema table provides all the ticket sales data needed for your analysis. To make it easier to see trends in large data sets, you need to visualize it graphically. In this section, you learn how to use **Power BI** to manipulate and visualize the tenant data you have extracted and 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.



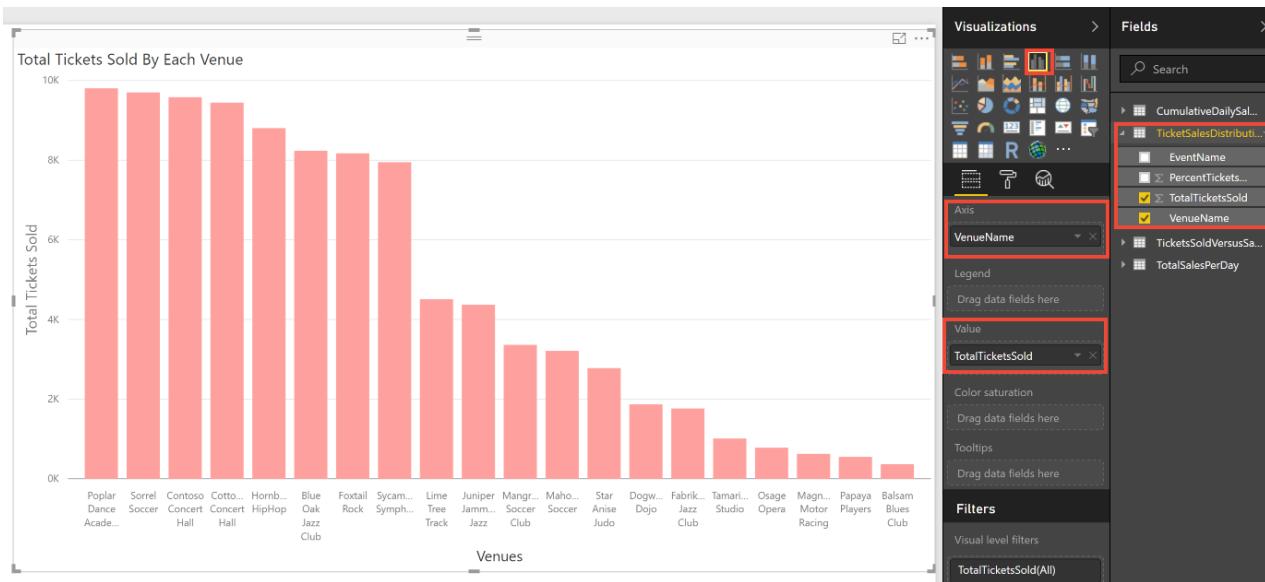
5. Select **Database** in the left pane, then enter user name = *developer*, and enter password = *P@ssword1*. Click **Connect**.



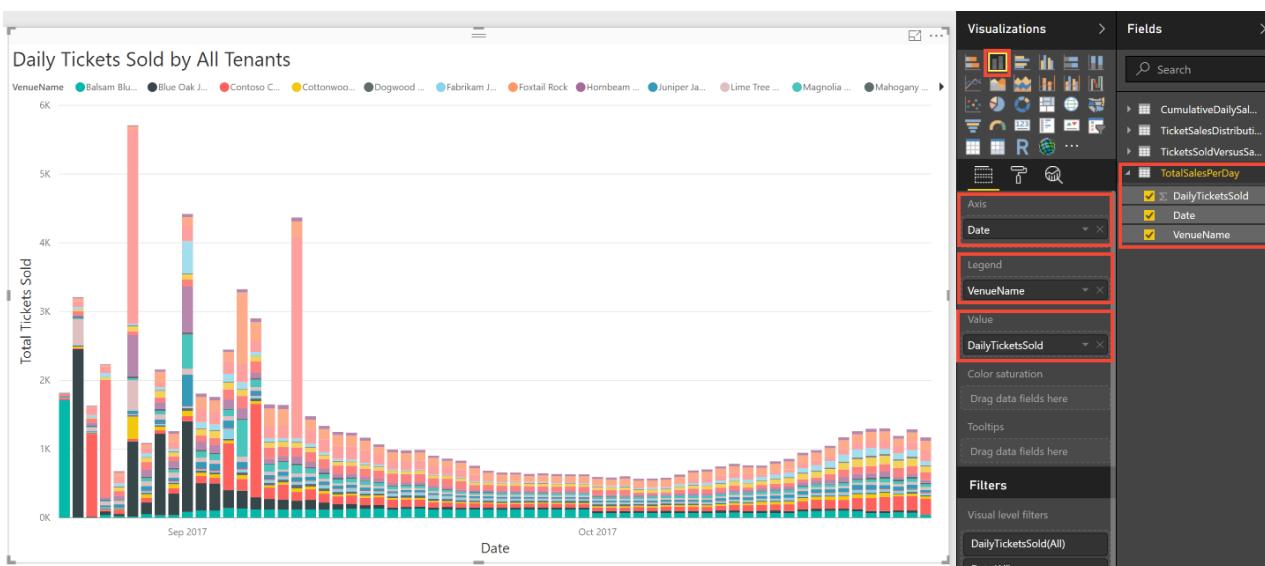
6. In the **Navigator** pane, under the analytics database, select the star-schema tables: fact\_Tickets, dim\_Events, dim\_Venues, dim\_Customers and dim\_Dates. Then select **Load**.

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.

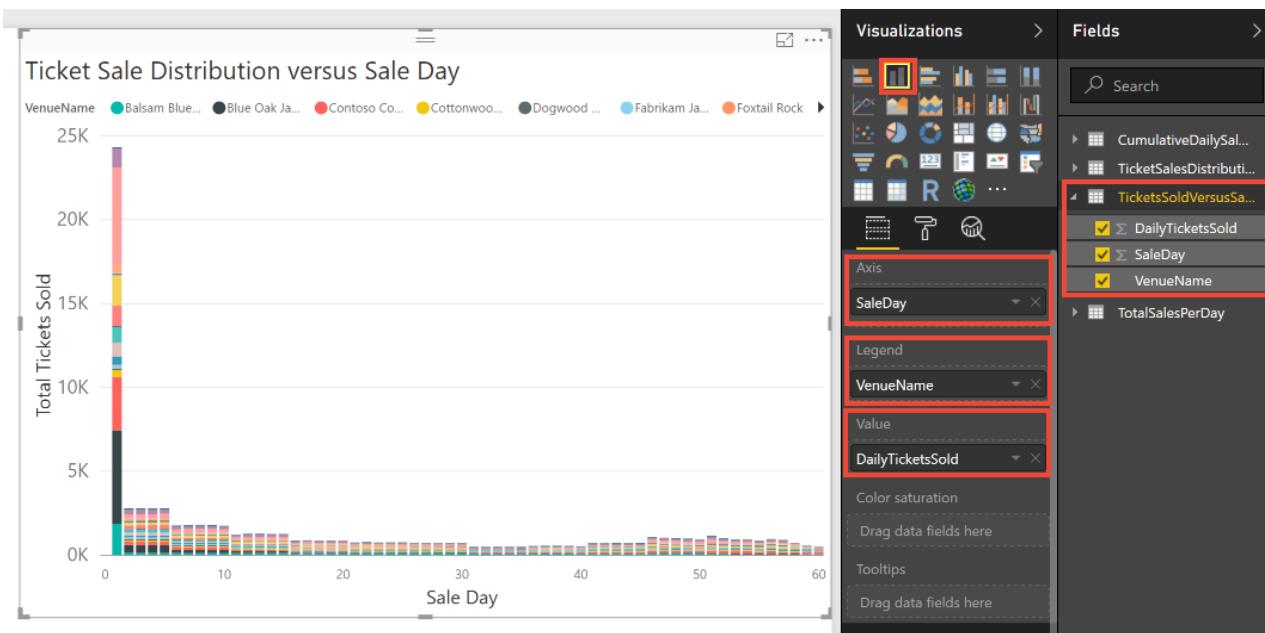
You start by analyzing ticket sales data to see the variation in usage across the venues. Select the following options in Power BI to plot a bar chart of the total number of tickets sold by each venue. Due to random variation in the ticket generator, your results may be different.



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.

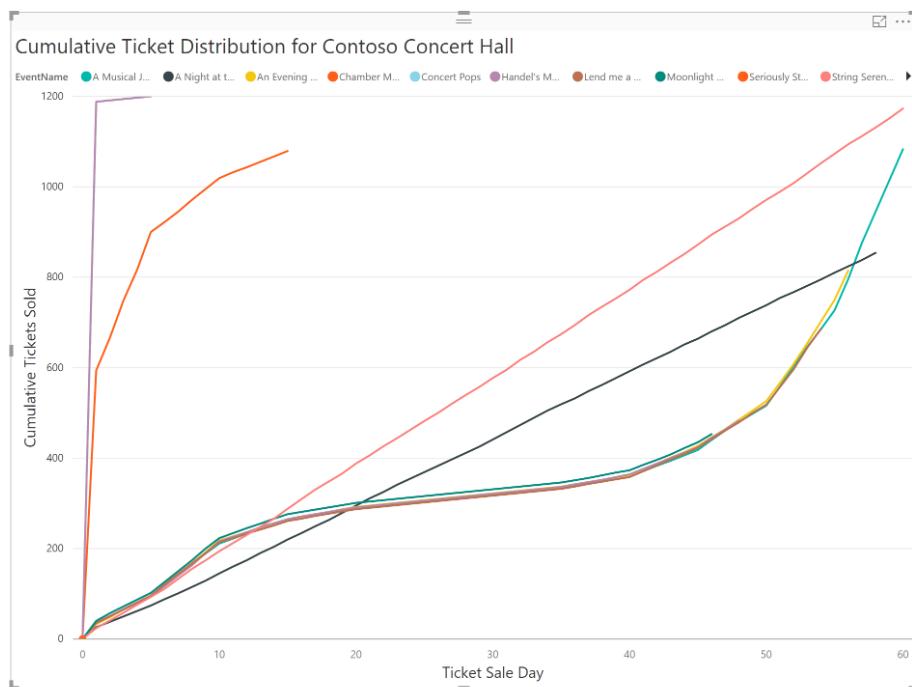


Next you want to further investigate the significance of these peak sale days. When do these peaks occur after tickets go on sale? To plot tickets sold per day, select the following options in Power BI.



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.

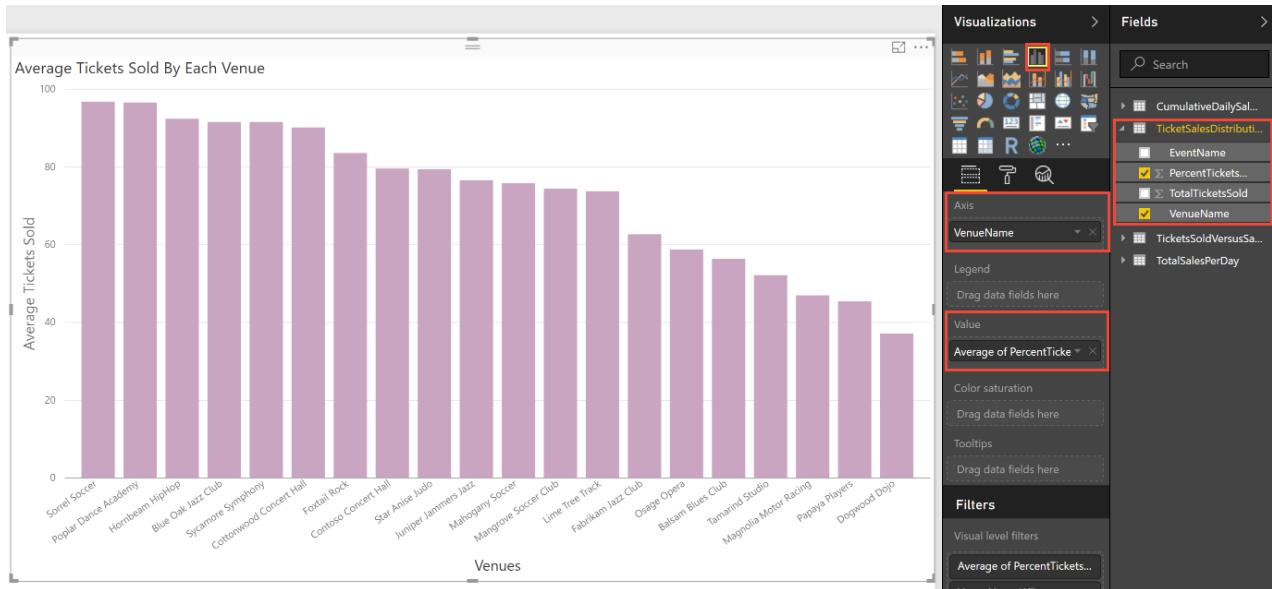
The insights into ticket selling patterns might lead Wingtip Tickets to optimize their business model. Instead of charging all tenants equally, perhaps Wingtip should introduce service tiers with different compute sizes. Larger venues that need to sell more tickets per day could be offered a higher tier 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](#)

# SQL Database frequently asked questions (FAQ)

10/29/2018 • 18 minutes to read • [Edit Online](#)

## What is the current version of SQL Database

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

## What is the SLA for SQL Database

We guarantee at least 99.99% of the time, you have connectivity between your Microsoft Azure SQL Database and our Internet gateway, regardless of your service tier. 0.01% is reserved for patches, upgrades, and failovers. For more information, see [SLA](#). For information about the availability architecture of Azure SQL Database, see [High Availability and Azure SQL Database](#).

## Can I control when patching downtime occurs

No. The impact of patching is generally not noticeable if you [employ retry logic](#) in your app.

## What is the new vCore-based purchasing model for Azure SQL Database

The new purchasing model is in addition to the existing DTU-based model. The vCore-based model is designed to give customers flexibility, control, transparency, and a straightforward way to translate on-premises workload requirements to the cloud. It also allows customers to scale their compute and storage resources based upon their workload needs. Single database and elastic pool options using the vCore model are also eligible for up to 30 percent savings with the [Azure Hybrid Benefit for SQL Server](#). See [DTU-based purchasing model](#) and [vCore-based purchasing model](#) for more information.

## What is a vCore

A virtual core represents the logical CPU offered with an option to choose between generations of hardware. Gen 4 Logical CPUs are based on Intel E5-2673 v3 (Haswell) 2.4-GHz processors and Gen 5 Logical CPUs are based on Intel E5-2673 v4 (Broadwell) 2.3-GHz processors.

## Is moving to the vCore-based model required

No, the introduction of the vCore-based model to the Elastic Pool and Single Database deployment options reflects our commitment to customer choice and flexibility. If customers want to continue using the DTU-based model, they don't need to take any action with this announcement and their experience and billing will remain unchanged.

In many cases, applications can benefit from the simplicity of a pre-configured bundle of resources. Therefore, we continue to offer and support these DTU-based choices to our customers. If you are using them and it meets your business requirements, you should continue to do so.

The DTU and vCore-based models will continue to exist side by side. We are launching the vCore-based model in response to customer requests for more transparency around their database resources and the ability to scale their compute and storage resources separately. The vCore-based model also enables additional savings for customers with active Software Assurance through the Azure Hybrid Benefit for SQL Server.

## How should I choose between the DTU-based purchasing model vs the vCore-based purchasing model

The Database Transaction Unit (DTU) is based on a blended measure of CPU, memory, reads, and writes. The DTU-based compute sizes represent preconfigured bundles of resources to drive different levels of application performance. Customers who do not want to worry about the underlying resources and prefer the simplicity of a preconfigured bundle while paying a fixed amount each month may find the DTU-based model more suitable for their needs. However, for customers who need more insight into the underlying resources or need to scale them independently to achieve optimal performance, the vCore-based model will be the best choice. Additionally, if a customer has an active Software Assurance (SA) for SQL Server, they can leverage their existing investment and save up to 30% with [Azure Hybrid Benefit for SQL Server](#). Options within each of the purchasing models provide the benefits of a fully-managed service such as automated backups, software updates and patches.

## What is the Azure Hybrid Benefit for SQL Server

The [Azure Hybrid Benefit for SQL Server](#) helps you maximize the value from your current licensing investments and accelerate their migration to the cloud. Azure Hybrid Benefit for SQL Server is an Azure-based benefit that enables you to use your SQL Server licenses with Software Assurance to pay a reduced rate ("base rate") on SQL Database. Azure Hybrid Benefit for SQL Server is available at public preview of the vCore-based purchasing model for SQL Database single databases and elastic pools. You may apply this benefit even if the SKU is active but note the base rate is applied from the time you select it in the Azure portal. No credit will be issued retroactively.

## 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, the SQL Server license can only be used in the cloud in SQL Database, and does not have dual use rights on-premises and in the cloud.

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

Today, we offer license mobility benefits to SQL Server customers with Software Assurance that allows re-assignment of their licenses to third-party shared servers. This benefit can be used 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 EE customers can get 4 cores in Azure in the General Purpose SKU for every core they own on-premises for highly virtualized applications. License mobility does not 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 is highly compatible with SQL Server on-premises

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

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

LICENSE FOOTPRINT

WHAT DOES AZURE HYBRID BENEFIT FOR SQL SERVER GET YOU?

| 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"> <li>• Can pay Base Rate on either General Purpose or Business Critical SKU</li> <li>• 1 core on-premises = 4 cores in General Purpose SKU</li> <li>• 1 core on-premises = 1 core in Business Critical SKU</li> </ul> |
| SQL Server Standard Edition core customers with SA   | <ul style="list-style-type: none"> <li>• Can pay Base Rate on General Purpose SKU only</li> <li>• 1 core on-premises = 1 core in General Purpose SKU</li> </ul>   |
|  |   |

## Is the vCore-based model available to SQL Database Managed Instance

[Managed Instance](#) is available only with the vCore-based model. For more information, also see the [SQL Database pricing page](#).

## Does the cost of compute and storage depend on the service tier that I choose

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 vCore price is approximately 2.7x higher in Business Critical. For the same reason, the higher storage price per GB in the Business Critical tier reflects the high IO and low latency of the SSD storage. At the same time, the cost of backup storage is not different because in both cases we use a class of standard storage.

## How am I charged for storage - based on what I configure upfront or on what the database uses

Different types of storage are billed differently. For data storage, you are charged for the provisioned storage based upon the maximum database or pool size you select. The cost does not 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. There's no additional charge for backup storage for up to 100 percent of your total provisioned server storage. Additional consumption of backup storage is charged in GB per month. For example, if you have the database storage size of 100 GBs, you'll get 100 GBs of backup at no additional cost. But if the backup is 110 GBs, you pay for the additional 10 GBs.

For backup storage of a single database, you are charged on a prorated basis for the storage that was allocated to the database backups minus the size of the database. For backup storage of an elastic pool, you are charged on a prorated basis for the storage that was allocated to the database backups of all the databases in the pool minus the maximum data size of the elastic pool. Any increase in the database size or elastic pool, or increase in the transaction rate, requires more storage and thus increases your backup storage bill. When you increase the maximum data size, this new amount is deducted from the billed backup storage size.

## How do I select the right SKU when converting an existing database to the new service tiers

For existing SQL Database applications using the DTU-based model, the General Purpose service tier is

comparable with the Standard tier. The Business Critical service tier is comparable with the Premium tier. In both cases, you should allocate at least 1 vCore for each 100 DTU that your application uses in the DTU-based model.

## Do the new vCore-based service tiers offer the compute sizes compatible with all existing compute sizes

The new vCore-based service tiers offer comparable performance choices for all elastic pools and databases using 100 DTUs or more. We will continue to add more compute sizes over time to accommodate sub 100 DTU workloads.

## Are there any database feature differences between the existing DTU-based and new vCore-based service tiers

The new service tiers support a superset of the features available with the current DTU-based offerings. The additional features include a set of additional dynamic management views (DMVs) and additional resource configuration options.

## If my database is CPU-bound and does not use much storage, can I increase the compute without paying for extra storage

Yes, you can independently select the level of compute your application needs and keep the storage unchanged. The storage can be set as low as 32GB.

## Will the new vCore-based tiers support point in time restore (PITR) for 35 days as today

You can configure the backup retention for PITR between 7 and 35 days. The backups storage will be charged separately based on the actual storage consumption if it exceeds the storage amount equal to the maximum data size. In preview, by default the PITR retention period is set to 7 days. In many cases, the maximum data size is sufficient for storing 7 days of backups.

## Why do you allow selection of the hardware generation for compute

Our goal is to enable maximum flexibility so that you can choose a performance configuration that closely matches the needs of the application. Gen4 hardware offers substantially more memory per vCore. However, Gen5 hardware allows you to scale up compute resources much higher. For more information, see [vCore purchase model](#)

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

The new service tiers offer a simple online conversion method that is similar to the existing process of upgrading databases from Standard to Premium service tier and vice versa. This conversion can be initiated using the Azure portal, PowerShell, Azure CLI, T-SQL, or the REST API. See [Manage single databases](#) and [Manage elastic pools](#).

## Can I convert a database from a vCore-based service tier to a DTU-based one

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

## Can I upgrade or downgrade between the General Purpose and Business Critical service tiers

Yes, with some restrictions. Your destination SKU must meet the maximum database or elastic pool size you configured for your existing deployment. If you are using [Azure Hybrid Benefit for SQL Server](#), the Business Critical SKU is only available to customers with Enterprise Edition licenses. Only customers who migrated from on-premises to General Purpose using Azure Hybrid Benefit for SQL Server with Enterprise Edition licenses can upgrade to Business Critical. For details see [What are the specific rights of the Azure Hybrid Benefit for SQL Server?](#)

This conversion does not result in downtime and can be initiated using Azure portal, PowerShell, Azure CLI, T-SQL, or the REST API. See [Manage single databases](#) and [Manage elastic pools](#).

## I am using a Premium RS database that will not be Generally Available - can I upgrade it to a new tier and achieve a similar price/performance benefit

Because the vCore model allows independent control over the amount of provisioned compute and storage, you can more effectively manage the resulting costs, making it an attractive destination for Premium RS databases. In addition, the [Azure Hybrid Benefit for SQL Server](#) provides a substantial discount when the vCore-based model is used.

## How often can I adjust the resources per pool

As often as you want. See [Manage elastic pools](#).

## How long does it take to change the service tier or compute size of a single database or move a database in and out of an elastic pool

Changing the service tier of a database and moving in and out of a pool requires the database to be copied on the platform as a background operation. Changing the service tier can take from a few minutes to several hours depending on the size of the databases. In both cases, the databases remain online and available during the move. For details on changing single databases, see [Change the service tier of a database](#).

## When should I use a single database vs. elastic databases

In general, elastic pools are designed for a typical [software-as-a-service \(SaaS\) application pattern](#), where there is one database per customer or tenant. Purchasing individual databases and over-provisioning to meet the variable and peak demand for each database is often not cost efficient. With pools, you manage the collective performance of the pool, and the databases scale up and down automatically. Azure's intelligent engine recommends a pool for databases when a usage pattern warrants it. For details, see [Elastic pool guidance](#).

## How does the usage of SQL Database using the DTU-based purchasing model show up on my bill

SQL Database bills on a predictable hourly rate based on the [purchasing model](#). Actual usage is computed and pro-rated hourly, so your bill might show fractions of an hour. For example, if a database exists for 12 hours in a month, your bill shows usage of 0.5 days.

## What if a single database is active for less than an hour or uses a higher service tier for less than an hour

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.

Examples:

- If you create a Basic database and then immediately upgrade it to Standard S1, you are charged at the Standard S1 rate for the first hour.
- If you upgrade a database from Basic to Premium at 10:00 p.m. and upgrade completes at 1:35 a.m. on the following day, you are charged at the Premium rate starting at 1:00 a.m.
- If you downgrade a database from Premium to Basic at 11:00 a.m. and it completes at 2:15 p.m., then the database is charged at the Premium rate until 3:00 p.m., after which it is charged at the Basic rates.

## How does elastic pool usage using the DTU-based purchasing model show up on my bill

Elastic pool charges show up on your bill as Elastic DTUs (eDTUs) or vCores plus storage in the increments shown on [the pricing page](#). 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.

DTU-based purchasing model examples:

- If you create a Standard elastic pool with 200 eDTUs at 11:18 a.m., adding five databases to the pool, you are charged for 200 eDTUs for the whole hour, beginning at 11 a.m. through the remainder of the day.
- On Day 2, at 5:05 a.m., Database 1 begins consuming 50 eDTUs and holds steady through the day. Databases 2-5 fluctuate between 0 and 80 eDTUs. During the day, you add five other databases that consume varying eDTUs throughout the day. Day 2 is a full day billed at 200 eDTU.
- On Day 3, at 5 a.m. you add another 15 databases. Database usage increases throughout the day to the point where you decide to increase eDTUs for the pool from 200 to 400 at 8:05 p.m. Charges at the 200 eDTU level were in effect until 8 pm and increases to 400 eDTUs for the remaining four hours.

## How are elastic pool billed for the DTU-based purchasing model

Elastic pools are billed per the following characteristics:

- An elastic pool is billed upon its creation, even when there are no databases in the pool.
- An elastic pool is billed hourly. This is the same metering frequency as for compute sizes of single databases.
- If an elastic pool is resized, then the pool is not billed according to the new amount of resources until the resizing operation completes. This follows the same pattern as changing the compute size of single databases.
- The price of an elastic pool is based on the resources of the pool. The price of an elastic pool is independent of the number and utilization of the elastic databases within it.

For details, see [SQL Database pricing](#), [DTU-based purchasing model](#), and [vCore-based purchasing model](#).

## How does the vCore-based usage show up in my bill

In the vCore-based model, the service is billed on a predictable, hourly rate based on the service tier, provisioned compute in vCores, provisioned storage in GB/month, and consumed backup storage. If the storage for backups exceeds the total database size (that is, 100% of the database size), there are additional charges. vCore hours, configured database storage, consumed IO, and backup storage are clearly itemized in the bill, making it easier for you to see the details of resources you have used. Backup storage up to 100% of the maximum database size is included, beyond which you are billed in GB/month consumed in a month.

For example:

- If the SQL database exists for 12 hours in a month, the bill shows usage for 12 hours of vCore. If the SQL database provisioned an additional 100 GB of storage, the bill shows storage usage in units of GB/Month prorated hourly and number of IOs consumed in a month.
- If the SQL database is active for less than one hour, you are billed for each hour the database exists using the highest service tier selected, provisioned storage, and IO that applied during that hour, regardless of usage or whether the database was active for less than an hour.
- If you create a Managed Instance and delete it five minutes later, you'll be charged for one database hour.
- If you create a Managed Instance in the General Purpose tier with 8 vCores, and then immediately upgrade it to 16 vCores, you'll be charged at the 16 vCore rate for the first hour.

**NOTE**

For a limited period, backup charges and IO charges are free of charge.

## How does the use of active geo-replication in an elastic pool show up on my bill

Unlike single databases, using [active geo-replication](#) with elastic databases doesn't have a direct billing impact. You are only charged for the resources provisioned for each of the pools (primary pool and secondary pool)

## How does the use of the auditing feature impact my bill

Auditing is built into the SQL Database service at no extra cost and is available on all service tiers. However, to store the audit logs, the auditing feature uses an Azure Storage account, and rates for tables and queues in Azure Storage apply based on the size of your audit log.

## How do I reset the password for the server admin

In the [Azure portal](#), click **SQL Servers**, select the server from the list, and then click **Reset Password**.

## How do I manage databases and logins

See [Managing databases and logins](#).

**NOTE**

You cannot change the name of the server admin account after it is created.

## How do I make sure only authorized IP addresses are allowed to access a server

See [How to: Configure firewall settings on SQL Database](#).

## What is an expected replication lag when geo-replicating a database between two regions within the same Azure geography

We are currently supporting an RPO of five seconds and the replication lag has been less than that when the geo-secondary is hosted in the Azure recommended paired region and at the same service tier.

## What is an expected replication lag when geo-secondary is created in

## the same region as the primary database

Based on empirical data, there is not too much difference between intra-region and inter-region replication lag when the Azure recommended paired region is used.

## If there is a network failure between two regions, how does the retry logic work when geo-replication is set up

If there is a disconnect, we retry every 10 seconds to re-establish connections.

## What can I do to guarantee that a critical change on the primary database is replicated

The geo-secondary is an async replica and we do not try to keep it in full sync with the primary. But we provide a method to 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](#).

## What tools are available to monitor the replication lag between the primary database and geo-secondary

We expose the real-time replication lag between the primary database and geo-secondary through a DMV. For details, see [sys.dm\\_geo\\_replication\\_link\\_status](#).

## To move a database to a different server in the same subscription

In the [Azure portal](#), click **SQL databases**, select a database from the list, and then click **Copy**. See [Copy an Azure SQL database](#) for more detail.

## To move a database between subscriptions

In the [Azure portal](#), click **SQL servers** and then select the server that hosts your database from the list. Click **Move**, and then pick the resources to move and the subscription to move to.

# Public data sets for testing and prototyping

10/1/2018 • 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 190,000 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, queriable, 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." |