

Introduction

Understanding Azure Database Administrator responsibilities, and how they relate to other Azure data platform roles, is important to performing the tasks and duties expected for this function.

In general, cloud services can be broken down into two sets of services: **Infrastructure as a Service (IaaS)** and **Platform as a Service (PaaS)**.

IaaS services typically consist of virtual machines, storage, and virtual networking components, and are largely managed by the user in terms of patching and software. Alternatively, PaaS services have a larger percentage of management tasks, which are handled by the cloud provider.

Learning objectives

At the end of this module, you will be able to:

- Understand the role of Azure Database Administrator, and other data platform roles
- Describe the key differences between the SQL Server-based database options in Azure
- Describe other features for Azure SQL platforms available

Describe Azure data platform roles

The role of Azure Database Administrator is just one of the roles that can be filled by professionals working with the Microsoft data platform services. There are five different role-based certifications offered by Microsoft for people whose main job responsibilities deal with cloud-focussed data.

Role	Definition
Azure Data Engineer	Design and implement the management, monitoring, security, and privacy of data using the full stack of Azure data services to satisfy business needs.
Azure Data Analyst	Enable businesses to maximize the value of their data assets by using Microsoft Power BI. As a subject matter expert, Data Analysts are responsible for designing and building scalable data models, cleaning and transforming data, and enabling advanced analytic capabilities that provide meaningful business value through easy-to-comprehend data visualizations.
Azure Data Scientist	Applies their knowledge of data science and machine learning to implement and run machine learning workloads on Azure; in particular, using Azure Machine Learning Service.
Azure Artificial Intelligence Engineer	Use Cognitive Services, Machine Learning, and Knowledge Mining to architect and implement Microsoft AI solutions involving natural language processing, speech, computer vision, bots, and agents.
Azure Database Administrator	Implements and manages the operational aspects of cloud-native and hybrid data platform solutions built on Microsoft Azure data services and Microsoft SQL Server. The Azure Database Administrator uses a variety of methods and tools to perform day-to-day operations, including applying knowledge of using T-SQL for administrative management purposes.

Understand SQL Server in an Azure virtual machine

A SQL Server running in an Azure virtual machine (IaaS) is equivalent to an on-premises SQL Server. You will notice that several features described for SQL Server on Azure virtual machine are applicable to all your on-premises SQL Servers.

Many applications will require SQL Server running on a virtual machine. The reasons include:

- **General application support and incompatibility** - For applications requiring an older version of SQL Server for vendor support. In addition, some application services may have a requirement to be installed with the database instance in a manner that is not compatible with a PaaS offering.
- **Use of other SQL Server Services** - In order to maximize licensing, many users choose to run SQL Server Analysis Services (SSAS), SQL Server Integration Services (SSIS), and/or SQL Server Reporting Services (SSRS) on the same machine as the database engine.

Versions of SQL Server available

Microsoft keeps images of all supported versions of SQL Server available in Azure Marketplace. If you have a need for an older version, that is covered by an extended support contract, you must install your own SQL Server binaries.

Backup solutions

In recent releases of SQL Server, Microsoft has introduced several features to support running SQL Server in an Azure virtual machine. **We are going to focus on two key backup features:**

- **Back up to URL**
- **Azure Backup**

Back up to URL allows you to use standard backup syntax to back up your databases to **Azure Blob Storage service**, while Azure Backup for SQL Server Virtual Machines offers a complete **enterprise backup solution that automatically handles your backups across your infrastructure**.

Deployment options

All resources in Azure share a common provider known as **Azure Resource Manager** that acts as a **management**, and deployment service for cloud services. While there are numerous ways to deploy Azure resources, ultimately, they all end up going into **JSON documents known as Azure Resource Manager template**, which is one of the deployment options for Azure resources.

The main difference between these processes is that Azure Resource Manager templates are a declarative deployment approach that describes the desired structure and state of the resources to be deployed, whereas the other methods can all be described as imperative, which uses procedural models to explicitly specify a process to be executed. In large-scale deployments, the declarative approach is better and should be followed.

Overview of Azure storage

Azure offers a fully redundant object-based storage model, and there are a few things to be aware of in designing and deploying Virtual Machines architecture. In terms of virtual machines, there are four types of storage you can use:

- Standard
- Standard SSD
- Premium SSD
- Ultra Disk

For production SQL Server data and transaction log files, you should only use Premium SSD storage and Ultra Disk. With premium storage, you will see latencies in the range of 5-10 ms on a properly configured system. Alternatively, with Ultra Disk you may have sub millisecond latency but will likely see 1-2 ms workloads in the real world. You can use Standard storage for your database backups, as the performance is adequate for most backup and restore workloads.

High availability in Azure

The Azure platform is designed to be **fault tolerant** and **provides quickly recovery from service disruptions and transient errors**. In fact, many organizations see higher levels of availability in single virtual machines deployments than they previously experienced in their on-premises environments. Microsoft guarantees uptime of **at least 99.9% for single instance Azure virtual machine, when using Premium SSD or Ultra Disk for all disks.**

Azure offers several features to support high availability including availability sets, availability zones, and load-balancing techniques that provide high availability by distributing incoming traffic among Virtual Machines.

Design Azure SQL Database for cloud-native applications

Azure SQL Database is a Platform as a Service (PaaS) that provides high scalability capabilities, and it can be a great solution for certain workloads, and requires minimal maintenance efforts.

Azure SQL Database is aimed at new application development as it gives developers a great deal of flexibility in building new application services, and granular deployment options at scale. SQL Database offers a low maintenance solution that can be a great option for certain workloads.

Purchasing model

SQL Database comes in two main purchasing models: **vCore-based model** and **DTU-based model**. Each purchasing model offers the following service tiers:

vCore-based

In this purchasing model, **compute and storage resources are decoupled**. It means you can scale storage and compute resources independently from one another. Here are listed the service tiers available:

Service tier	Capability
General Purpose	This service tier is designed for less intensive operations, and offers budget oriented balanced compute and storage options. It provides both provisioned compute tier and serverless compute tier.
Business Critical	This service tier supports In-Memory OLTP, and built-in read-only replica. It also includes more memory per core, and uses local SSD storage, which is designed for performance-sensitive workloads.
Hyperscale	Hyperscale introduces horizontal scaling features that use advanced techniques to add compute nodes as the data sizes grow. It is only supported on single SQL database . Hyperscale allows you to scale storage and compute resources significantly over the limits available for the General Purpose and Business Critical service tiers.

DTU-based

In the DTU model, there are three service tiers available: **Basic, Standard, and Premium**. Compute and storage resources are dependent on the DTU level, and they provide a range of performance capabilities at a fixed storage limit, backup retention, and cost.

For example, if your database grows to the point it reaches the maximum storage limit, you would need to increase your DTU capacity, even if the compute utilization is low.

The scaling operation on SQL Database may incur in a brief connection interruption at the end of the scaling operation. There are two main changes that trigger this behavior:

- Once you initiate a scaling operation that requires an internal failover.
- When adding or removing databases to the elastic pool.

You can use a proper **retry logic** in your application to handle connection errors.

NOTE: Azure SQL Managed Instance doesn't support DTU-based purchasing model.

Serverless compute tier

Despite its name, the serverless compute tier does require you to have a server with your database. **The serverless option can best be thought of as an autoscaling and auto-pause solution for SQL Database. It is effective for lowering the costs in development and testing environments.** For example, you can set up a minimum and a maximum vCores configuration for your database, where it will scale dynamically based on your workload.

The auto-pause delay feature allows you to define the period of time the database will be inactive before it is automatically paused. The auto-pause delay feature can be set up from 1 hour to seven days. Alternatively, auto-pause delay feature can be disabled.

The resume operation is triggered when the next attempt to access the database occurs, and only **storage charges are applicable** when the database is paused.

Configure ...

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

Service tier: General Purpose (Scalable compute and storage options) [Compare service tiers](#)

Compute tier: **Serverless** - Compute resources are auto-scaled. Billed per second based on vCores used.

Compute Hardware

Select the hardware configuration based on your workload requirements. Availability of compute optimized, memory optimized, and confidential computing hardware depends on the region, service tier, and compute tier.

Hardware Configuration: Gen5
up to 40 vCores, up to 120 GB memory [Change configuration](#)

Max vCores: 4 vCores

Min vCores: 0.5 vCores

2.1 GB MIN MEMORY 12 GB MAX MEMORY

Auto-pause delay: The database automatically pauses if it is inactive for the time period specified here, and automatically resumes when database activity recurs. Alternatively, auto-pausing can be disabled.
 Enable auto-pause
Days: 0 Hours: 1 Minutes: 0

Data max size (GB): 32
9.6 GB LOG SPACE ALLOCATED

Cost summary

Gen5 - General Purpose (GP_S_Gen5_4)
Cost per GB (in USD)
Max storage selected (in GB) x 41.6

ESTIMATED STORAGE COST / MONTH	USD
COMPUTE COST / VCORE / SECOND ¹	USD

NOTES
¹ Serverless databases are billed in vCores based on a combination of CPU and memory utilization. Learn more about serverless billing

The image above shows where you can change the autoscaling and auto-pause properties for serverless compute tier.

Deployment model

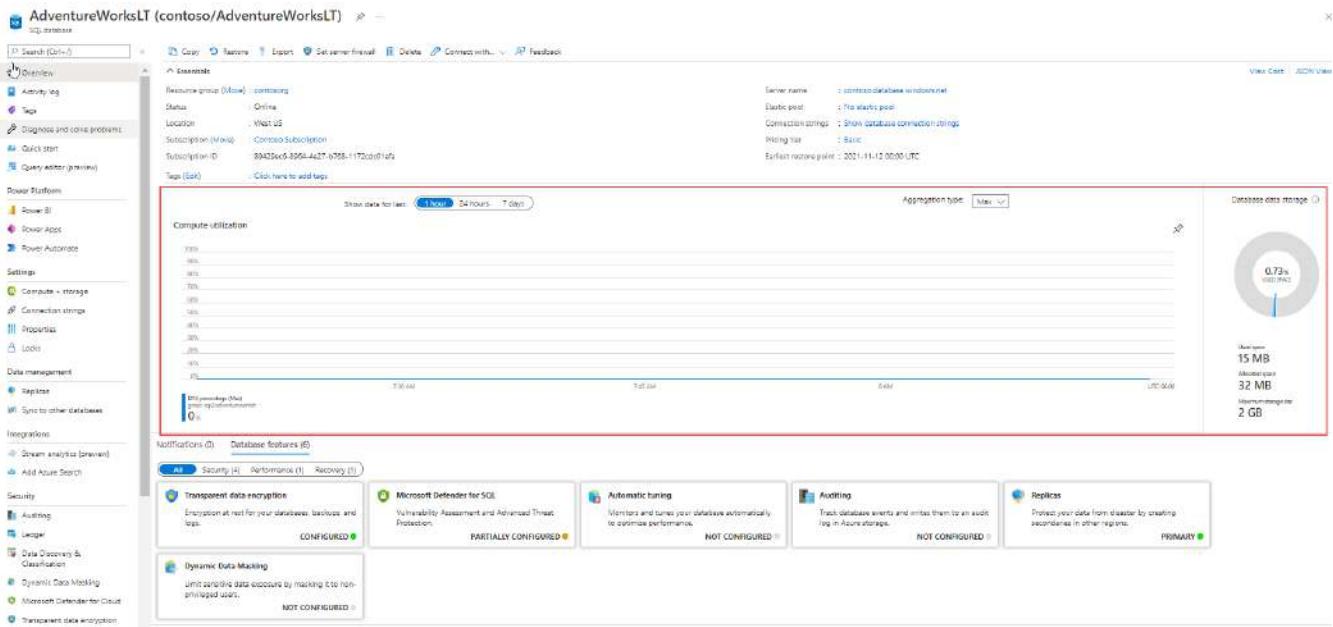
There are two main deployment models when deploying SQL Database on Azure: **single database** and **elastic pool**. Elastic pools share resources with other databases part of the same pool, while single databases resources are managed independently.

SQL Database, like virtual machines, can be deployed using Azure Resource Manager templates, PowerShell, Azure CLI, or the Azure portal.

Single database

Single database is the simplest deployment model of Azure SQL Database. You manage each of your databases individually from scale and data size perspectives. **Each database deployed in this model has its own dedicated resources, even if deployed to the same logical server.**

You can monitor database resources utilization through Azure portal. This feature allows you to easily identify how the database is performing, as shown in the image below:



Elastic pool

Elastic pools allow you to allocate storage and compute resources to a group of databases, rather than having to manage resources for each database individually. Additionally, **elastic pools are easier to scale than single databases, where scaling individual databases is no longer needed due to changes made to the elastic pool.**

Elastic pools provide a cost-effective solution for software as a service application model, since resources are shared between all databases. You can configure resources based either on the DTU-based purchasing model or the vCore-based purchasing model.

Due to the nature of this feature, it is recommended to monitor your resources continually to identify concurrent performance spikes that could affect other databases part of the same elastic pool. Often, you may need to revisit your allocation strategy to make sure there's enough resource available for all databases sharing the same elastic pool.

Elastic pool is a good fit for multi-tenant architecture with low average utilization, where each tenant has its own copy of the database.

Network options

Azure SQL Database by default has a public internet endpoint. Access to this endpoint can be controlled via firewall rules, or limited to specific Azure networks, using features like Virtual Network endpoints or Private Link.

Backup and restore

Azure provides seamless backup and restore capabilities for SQL Database and SQL Managed Instance. Let's learn about some of the most important features.

Continuous backup

With SQL Database, you can increase administration efficiency by knowing that databases are backed up regularly, and that they are continuously copied to a read-access geo-redundant storage (RA-GRS).

Full backups are performed every week, differential backups every 12 to 24 hours, and transaction log backups every 5 to 10 minutes.

Geo-restore

As backups are geo-redundant by default for SQL Database and SQL Managed Instance, you can easily restore databases to a different geographical region, which is especially useful for less strict disaster recovery scenarios.

Backup storage is billed apart from regular database files storage. However, when provisioning a SQL Database, the backup storage is created with the maximum size of the data tier selected for your database at no extra cost.

The duration of a geo-restore operation can be affected by several underlying components including the size of the database, the number of transaction logs involved in a restore operation, and the amount of simultaneous restore requests being processed in the target region.

NOTE: Geo-restore is available when the backup storage redundancy property is set to *geo-redundant backup storage*.

Point-in-time restore (PITR)

You can configure a specific point in time retention policy for each database running on a SQL Database offering. SQL Database retention period can be set from 1 up to 35 days. In fact, if not specified, the default configuration is seven days.

You can restore your databases to a specific point in time according to the retention defined, but **PITR is only supported if you are restoring a database in the same server the backup was originated**. You can use either Azure portal, Azure PowerShell, Azure CLI, or REST API to restore a SQL Database.

Long-term retention (LTR)

Long-term retention is useful for scenarios that require you to set the retention policy beyond what is offered by Azure. **You can set a retention policy for up to 10 years**, and this option is **disabled by default**.

The screenshot shows the Azure portal interface for managing a database's retention policies. On the left, there's a sidebar with navigation links like Home, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Settings, Data management (with Backup selected), Security, and Intelligent Performance. The main content area shows a table of retention policies for the 'AdventureWorksLT' database. The 'Retention policies' tab is highlighted. A red box highlights the 'Configure policies' button at the top right of the main content area. To the right, a modal window titled 'Configure policies' is open, divided into sections: 'Point-in-time restore' (specifying 7 days for PTR), 'Differential backup frequency' (set to 12 hours), and 'Long-term retention' (with weekly, monthly, and yearly backup options). Buttons for 'Apply' and 'Cancel' are at the bottom of the modal.

In the image above, you can configure long-term retention policies through Azure portal. Once the database is selected, a new panel will open on the right side of the screen, where you can override the default properties.

For more information about automated backups, see [Automated backups - Azure SQL Database & Azure SQL Managed Instance](#).

Automatic Tuning

Automatic tuning is a built-in feature that relies on machine learning regression capabilities, and automatically identify tuning opportunities based on your query performance.

Automatic tuning currently includes the following features:

- Identify Expensive Queries
- Forcing Last Good Execution Plan
- Adding Indexes
- Removing Indexes

The Azure services use a combination of built-in advanced features to determine the best indexes for your query pattern. Initially, these indexes are tested on a copy of your database, and finally applied to your database.

All databases inherit configuration from their parent server, and you can easily disable this feature at any time.

Elastic query (preview)

Elastic query allows you to run T-SQL queries that bridge multiple databases in SQL Database. This feature is particularly useful for applications using three- and four-part names that cannot be changed. It also increases portability as it allows for migration.

Elastic queries support the following partitioning strategies:

Service tier	Capability
Vertical partitioning	Also called cross-database queries. The data is partitioned vertically between many databases. Schemas are different for each database. For example, you may have a database used for customer data, and a different one used for payment information. With the help of vertical partitioning, you can now run a cross-database query between both databases.
Horizontal Partitioning	Also called sharding. The data is partitioned horizontally to distribute rows across several scaled-out databases. In this topology, the schema remains the same among all sharding databases. It supports either single-tenant model or multiple tenant models.

NOTE: Azure SQL Managed Instance doesn't support elastic queries.

Elastic job (preview)

The elastic job feature is the SQL Server Agent replacement for Azure SQL Database. To some extent, elastic job is equivalent to the Multi Server Administration feature available on an on-premises SQL Server instance.

You can execute T-SQL commands across several target deployments like SQL Databases, SQL Database elastic pools, and SQL Databases in shard maps. Database resources can run on different Azure subscriptions, and/or regions. The execution runs in parallel, which is useful when automating database maintenance tasks.

NOTE: Azure SQL Managed Instance doesn't support elastic jobs.

SQL Data Sync

The Data Sync feature allows you to incrementally synchronize data across multiple databases running on SQL Database or on-premises SQL Server. Similarly, Data Sync is a good option if you need to offload intensive workloads in production with a separate database that can be used for analytics and/or ad hoc operations.

Data Sync is based on a hub topology, where you define one of the databases in the sync group to work as a hub database. The sync group can have multiple members, and you can only synchronize changes between the hub database and individual databases. **Data Sync tracks changes using insert, update, and delete triggers through a historical table created on the user database.**

When you create a sync group, it will ask you to provide a database responsible to store the sync group metadata. The metadata location can be either a new database or an existing database as long it resides in the same region as your sync group.

Create Data Sync Group

...

Create a sync group on the hub database. Member databases and other additional settings can be configured after create in the sync group details blade. [Learn more](#)

Sync Group Name *

 ✓

Sync Metadata Database

Use existing database New database

 ✓

Automatic Sync *

On Off

Conflict Resolution *

 ✓

Use private link

In the image above, you can specify sync group properties like the schedule synchronization, the conflict resolution option, and the use of a private link if needed.

NOTE: Azure SQL Managed Instance doesn't support Data Sync feature.

For more information about how to configure SQL Data Sync, see [Tutorial: Set up SQL Data Sync between databases in Azure SQL Database and SQL Server](#).

Explore Azure SQL Database Managed Instance

Most of the features available on Azure SQL Database will also work for Azure SQL Managed Instance as they share the same base code. The fully managed platform as a service (PaaS) provides some of the following benefits:

- Automatic backups
- Automatic patching
- Built-in high availability
- Security and performance tools
- Embedded auditing capabilities

Another key benefit when migrating to one of the PaaS offerings on Azure is that you no longer have to install or patch SQL Server, which can increase application uptime, and reduce maintenance efforts.

Unlike Azure SQL Database, which is designed around single database structures, SQL Managed Instance provides several other features including cross-database queries, common language runtime (CLR), access to the system databases, and use of the SQL Agent features.

For a complete list of the features available on Azure SQL Managed Instance, see [Features of SQL Database and SQL Managed Instance](#).

Hybrid licensing options

Microsoft offers several benefits to SQL Server licenses. For both SQL Database and SQL Managed Instance, taking advantage of your existing licenses can reduce the cost of running the PaaS offering.

- For each core of Enterprise Edition with Active Software Assurance, you're eligible for one vCore of SQL Database or SQL Managed Instance Business Critical, and eight vCores of General Purpose.
- For each core of Standard Edition with Active Software Assurance, you're eligible for one vCore of General Purpose.

This model can reduce the total license costs by up to 40%. Effectively, you'll only be paying for the compute and storage costs, and not the software licensing costs.

For more information on bring-your-own licensing model, see [License Mobility through Software Assurance on Azure](#).

Connectivity architecture

Connections to SQL Managed Instance are made through TDS endpoints. While the routing and security on these connections differ, there's a gateway component that handles and routes connections to the database service. This gateway component is also deployed in a highly available fashion.

Backup and restore

Automated database backup provides a fully managed backup service that takes full, differential, and log backups regularly for both SQL Managed Instance and SQL Database offerings. Automated backups are geo-redundant, and replicated to a paired-region automatically, which protects your data against localized outages in the primary region.

Similarly, SQL Managed Instance allows easy migration of existing applications, enabling restores from on-premises backups.

There are some important considerations when running backup and restore operations on SQL Managed Instance databases:

- It isn't possible to overwrite an existing database during the restore process. Before restoring a database, you must ensure that it doesn't exist.
- For SQL Managed Instance, backups can only be restored to another managed instance. It isn't possible to restore a managed instance database backup to a SQL Server running on a virtual machine or SQL Database.
- Copy-only backup to Azure blob storage is available for SQL Managed Instance. SQL Database doesn't support this feature.

For more information about automated backups, see [Automated backups - Azure SQL Database & Azure SQL Managed Instance](#).

High availability architecture

SQL Database and SQL Managed Instance have similar high availability architectures, which guarantee 99.99% uptime. Windows and SQL Server updates are handled by the backend infrastructure, generally without any effect to your application, though it's important to place a [retry logic](#) into your application.

The **auto-failover groups feature** allows you to fail over a group of replicated databases on a server to another region. This feature is designed on top of the existing active geo-replication capability, which simplifies deployment and management of geo-replicated databases.

A failover group can include one or multiple databases, often used by the same application. Additionally, you can use the readable secondary databases to offload read-only query workloads.

NOTE: The auto-failover groups feature is supported on both SQL Managed Instance and SQL Database.

For more information about auto-failover groups, see [Use auto-failover groups to enable transparent and coordinated geo-failover of multiple databases](#).

Migration options

In general, migrating to SQL Managed Instance is often simple given the large set of available features. There are a couple of ways to migrate on-premises databases:

- Restoring a backup
- Using Database Migration Service (DMS)

Backup and restore will incur more downtime, as it isn't possible to restore with NORECOVERY option, and apply log backups.

The Database Migration Service is a managed service that connects your on-premises (or Azure Virtual Machines) SQL Server to SQL Managed Instance with near zero downtime. As a result, it acts like an automated log shipping process, meaning you can keep your target databases in sync right up to the point of cutover.

Machine Learning Services

Machine Learning Services provides machine learning operations within your relational database structure. This feature supports Python and R packages, ideal for high-intensive predictive capabilities. This option is available on SQL Managed Instance, SQL Server on Azure virtual machine, and on-premises SQL Server.

Applications can use relational database on Azure combined with machine learning high-performance capabilities, where you can:

- Train machine learning models based on either sampled dataset or population dataset.
- Reduce complexity in security and compliance, where you don't need to relocate your data to build and train your machine learning models.
- Deploy machine learning models using T-SQL stored procedures that support Python or R programming language.
- Use of open-source libraries like scikit-learn, PyTorch, and TensorFlow.

For busy environments, you can use the **T-SQL PREDICT function**, which allows you to accelerate predictions based on your stored model.

The Machine Learning Services feature can be enabled by running the following command:

```
EXEC sp_configure 'external scripts enabled', 1;
RECONFIGURE WITH OVERRIDE;
```

The command above allows the execution of external scripts in your managed instance, and it should be enabled before you attempt to use **sp_execute_external_script** to execute Python or R scripts in your database.

NOTE: SQL Database doesn't support Machine Learning Services feature.

For more information about Machine Learning Services, see [Machine Learning Services in Azure SQL Managed Instance](#).

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What Azure storage is recommended for production data files that requires maximum throughput?

- Premium SSD
- Ultra SSD
- Standard SSD

Check Answers

Multiple choice

What option should you select to change the auto-pause setting of a serverless SQL Database?

- The auto-pause setting can't be changed, and it's fixed at one hour
- Ensure that there are no active transactions in the database, as the auto-pause change will result in a disconnection
- The Azure portal allows you to change the auto-pause setting at any time

Check Answers

Multiple choice

Which SQL Server PaaS offering should you choose if you want to restore an on-premises database backup to Azure?

- Azure SQL Managed Instance
- Azure SQL Database
- You can use both Azure SQL Database and Azure SQL Managed Instance.

Check Answers

Summary

Microsoft provides a great deal of options for deploying SQL Server to Azure. Migrating on-premises databases to Azure SQL Server on Virtual Machine offers portability, while Azure SQL Database and Azure SQL Managed Instance provide scalability and flexibility.

Now that you've reviewed this module, you should be able to:

- Understand the role of Azure Database Administrator, and other data platform roles
- Describe the key differences between the SQL Server-based database options in Azure
- Describe other features for Azure SQL platforms available

Introduction

The most common reason for deploying SQL Server in an Azure Virtual Machine (VM) is because you want an easy, straightforward method to migrate an existing on-premises SQL Server into the cloud.

Understanding the options and methods for deploying SQL Server in an Azure VM is critical to ensure a successful migration. Infrastructure as a Service (IaaS) allows for greater flexibility in configuration. This flexibility means that you, as a database administrator, must plan your configuration carefully. **Choosing the proper VM sizing, storage, and networking options is crucial to ensure adequate performance for your workloads.**

Learning objectives

At the end of this module, you will be able to:

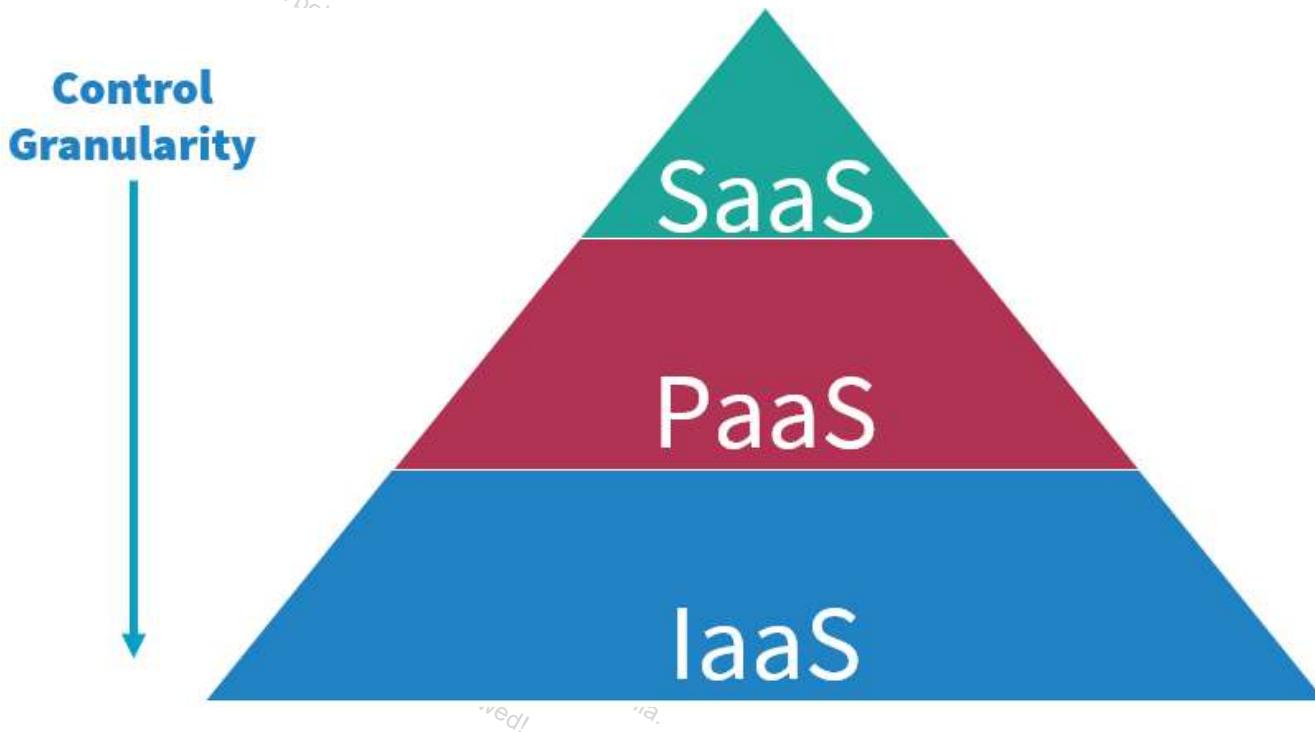
- Explore the basics of SQL Server in an Infrastructure as a Service (IaaS) offering
- Learn how hybrid scenario works
- Explore performance and security options available
- Understand the high availability and disaster recovery options

Explain IaaS options to deploy SQL Server in Azure

Many applications will require a VM running SQL Server. Some reasons for this option include:

- **Older versions of SQL Server**—If an application requires an older version of SQL Server for vendor support, running inside a VM is the best option for those applications, because it allows for the application to be supported by that vendor.
- **Use of other SQL Server services**—While Analysis Services and to an extent Integration Services (through the use of Azure Data Factory) are available as PaaS offerings, many users maximize their licensing by running SQL Server Analysis Services, Integration Services, or Reporting Services on the same machine as the database engine.
- **General application incompatibility**—This reason is a somewhat of a catch-all. **For example, Azure SQL Database doesn't support cross-database querying, while managed instance does.** Some applications may require additional services to be co-located with the database instance in a manner that isn't compatible with a PaaS offering.

Infrastructure as a Service (IaaS) allows the administrator to have more granular access over specific settings of the underlying infrastructure than the other Azure offerings. While the Azure platform manages the underlying server and network hardware, you still have access to the virtual storage, virtual networking configuration, and any additional software you might install within the virtual machine. This includes Microsoft SQL Server.



The image above illustrates the increased control you have using IaaS, compared to the other Azure SQL offerings. **While the exact configuration options vary between service offerings, typically in SaaS offerings the administrator is responsible only for user security and possibly data management. When using PaaS services, the operating system (OS) and other software are managed by the cloud provider.** A good example of this is the Azure Database platform where the operating system and RDBMS are installed and configured by Microsoft, allowing you to start building database applications quickly. IaaS solutions are the most open ended; you are responsible for OS patching, as well as optimal configuration of your network and storage options. With an IaaS deployment, you are also responsible for software configuration.

For IaaS solutions running in Azure, Microsoft will manage any resource below the operating system, including the physical servers, storage and physical networking. The database administrator is responsible for configuration of your SQL Server instances running on the operating system.

Some of your applications may not be suited for other Azure offerings, such as Azure SQL Database, because they require specific operating conditions. These conditions could include a specific combination of SQL Server and Windows versions for vendor support purposes, or additional software that needs to be installed alongside of SQL Server. SQL Server paired with the Azure IaaS platform provides the required control options for many organizations, whether it be specific feature like CLR or replication, or the use of Active Directory (as opposed to Azure Active Directory) authentication. Another requirement is that some applications install software alongside SQL Server, which requires direct access to the underlying operating system. Direct access to the OS isn't supported in a PaaS model. These organizations and their applications can obtain the advantages of moving to a cloud service without losing critical capabilities that their organization requires.

SQL Server IaaS Agent Extension

When you deploy an SQL Server VM from the Azure Marketplace, part of the process installs the IaaS Agent Extension.

The screenshot shows the Azure Marketplace page for the 'SQL Server 2019 on Windows Server 2019' offering. At the top, there's a red icon with a monitor and the word 'SQL'. Below it, the title 'SQL Server 2019 on Windows Server 2019' is displayed, along with the Microsoft logo. A 'Save for later' button with a heart icon is visible. Underneath, there's a section titled 'Select a software plan' with a dropdown menu set to 'SQL Server 2019 Web on Windows ...'. There are two buttons: 'Create' (in blue) and 'Start with a pre-set configuration'. Below these buttons is a link 'Deploy with Resource Manager (change to Classic)'. At the bottom of the main content area, there are two tabs: 'Overview' (which is underlined in blue) and 'Plans'. The 'Overview' tab contains links to 'Useful Links', 'Documentation', 'SQL Server 2019 information', 'Support forum', and 'Pricing details'.

Extensions are code that is executed on your VM post-deployment, typically to perform post deployment configurations. Some examples are installing anti-virus features, or installing a Windows feature. The SQL Server IaaS Agent Extension provides the following main features that can reduce your administrative overhead.

- **Automated backup**
- **Automated patching**
- **Azure Key Vault integration**
- **Defender for Cloud portal integration**
- **View Disk utilization in the portal**
- **Flexible licensing**

- **Flexible version or edition**
- **SQL best practices assessment**

In addition to these features, the extension allows you to view information about your SQL Server's configuration and storage utilization.

The screenshot shows the Azure portal interface for a SQL virtual machine named 'Win1'. The left sidebar includes links for Overview, Access control (IAM), Settings (Configure, Security, Patching, Backups, Additional features, Properties), Support + troubleshooting (New support request), and Home > Win1. The main content area displays the following details:

Setting	Value
Resource group (change)	: BookDemo
Status	: Online
Location	: East US 2
Subscription (change)	: Visual Studio Ultimate with MSDN
Subscription ID	: [REDACTED]
SQL configuration	: Click here to view SQL extension configuration
Tags (change)	: Click here to add tags
Version	: SQL Server 2019
Edition	: Developer
Virtual machine	: Win1
Virtual machine OS	: Windows (Windows Server 2019 Datacenter)
Virtual machine size	: Standard D2s v3
License type	: Pay As You Go

Below this, there is a storage utilization chart showing disk usage across drives G, C, F, and D. The chart indicates the following approximate values:

Drive	Type	Size
G	SQL data	46 MiB
C	System Reserved	47 MiB
F	SQL log	24.2 GiB
D	Available	2.1 TiB

On the right side, there is a 'Features' section with seven items, all of which are marked as 'CONFIGURED' with green status indicators:

- License type**: Configure the licensing model and SQL server edition for the SQL Server virtual machine.
- Storage configuration**: Automate your storage configuration by attaching storage to the VM, making that storage accessible to SQL Server.
- SQL connectivity**: Configuration the connection to the SQL Server instance running on your Azure virtual machine.
- Azure Key Vault integration**: Configure the Azure Key Vault (AKV) service to manage and store the cryptographic keys.
- Automated patching**: Establish a maintenance window for an Azure virtual machine.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Resource group (change) : BookDemo

Status : Online

Location : East US 2

Subscription (change) : Visual Studio Ultimate with MSDN

Subscription ID :

SQL configuration : Click here to view SQL extension configuration

Tags (change) : Click here to add tags

Version : SQL Server 2019

Edition : Developer

Virtual machine : Win1

Virtual machine OS : Windows (Windows Server 2019 Datacenter)

Virtual machine size : Standard D2s v3

License type : Pay As You Go

Notifications (0) Features (7)

All Pricing (2) Security (2) Configuration (3)

License type Configured
Configure the licensing model and SQL server edition for the SQL Server virtual machine.

Storage configuration Configured
Automate your storage configuration by attaching storage to the VM, making that storage accessible to SQL Server.

SQL connectivity Configured
Configuration the connection to the SQL Server instance running on your Azure virtual machine.

Azure Key Vault integration Not Configured
Configure the Azure Key Vault (AKV) service to manage and store the cryptographic keys.

Automated patching
Establish a maintenance window for an Azure VM

SQL Server licensing models

There are several different options related to how SQL Server is licensed when using the Azure IaaS offering.

If you aren't participating in the Microsoft Software Assurance (SA) program, you can deploy an image from the Azure Marketplace containing a pre-configured SQL Server, and **pay-per-minute for the use of SQL Server**. This option is referred to as the Pay as you Go model and the cost of the SQL Server license is included with the cost of the virtual machine.

If you are participating in the Microsoft Software Assurance (SA) program, you have more flexibility in how you license your SQL Server:

- You can use the previous method and pay-per-minute by deploying a virtual machine image containing a SQL Server from the Azure Marketplace
- You can Bring Your Own License (BYOL) when deploying the virtual machine that doesn't contain a pre-configured SQL Server instance. This option is possible when you already have purchased a valid SQL Server license for your on-premises infrastructure. This license can be applied to the virtual machine to ensure that you're properly licensed. You must report the usage of licenses to Microsoft by using the License Mobility verification form within 10 days of implementing the virtual machine.

When choosing this method, you can manually install SQL Server through media you have obtained, or you can choose to upload a virtual machine image to Azure.

In addition to flexible licensing options for SQL Server, there are also Windows Server licensing options that can be taken advantage of. These Windows Server options are known as the Azure Hybrid Benefit (AHB). Similar to applying a SQL Server license you already have purchased, you're able to take advantage of Windows Server licenses you already own.

Reserving a virtual machine for one to three years provides another option for cost savings. This commitment doesn't require an upfront payment and can be billed monthly. Using the reservation option can be beneficial if you know the workloads are going to be persisted. The cost savings can be significant, especially for larger VMs.

Virtual machine families

When deploying to an Azure virtual machine, there are several series, or “families”, of virtual machine sizes that can be selected. Each series is a combination of memory, CPU, and storage that meets certain requirements. For example, the series that are compute optimized have a higher CPU to memory ratio. Having multiple options allows you to select an appropriate hardware configuration for the expected workload. The following six series each have various sizes available, the details of which are fully described in the Azure portal when you choose the option to select your VM size.

General purpose - These VMs provide a balanced ratio of CPU to memory. This VM class is ideal for testing and development, small to medium-sized database servers, and web servers with a low to medium amount of traffic.

Compute optimized - Compute optimized VMs have a high CPU-to-memory ratio and are good for web servers with a medium amount of traffic, network appliances, batch processes, and application servers. These VMs can also support machine learning workloads that can't benefit from GPU-based VMs.

Memory optimized - These VMs provide high memory-to-CPU ratio. These VMs cover a broad range of CPU and memory options (all the way up to 4 TB of RAM) and are well suited for most database workloads.

Storage optimized - Storage optimized VMs provide fast, local, NVMe storage that is ephemeral. They are good candidates for scale-out data workloads such as Cassandra. It is possible to use them with SQL Server, however since the storage is ephemeral, you need to ensure you configure data protection using a feature like Always On Availability Groups or Log Shipping.

GPU - Azure VMs with GPUs are targeted at two main types of workloads—naturally graphics processing operations like video rendering and processing, but also massively parallel machine learning workloads that can take advantage of GPUs.

High performance compute - High Performance Compute workloads support applications that can scale horizontally to thousands of CPU cores. This support is provided by high-performance CPU and remote direct memory access (RDMA) networking that provides low latency communications between VMs.

The easiest way to see the sizing options within each series is through the Azure portal. From the blade for creating a VM, you can click the option to “Select Size” and see a list.

Select a VM size								
Browse available virtual machine sizes and their features								
VM Size ↑↓	Offering ↑↓	Family	vCPUs ↑↓	RAM (GiB) ↑↓	Data disks ↑↓	Max IOPS	↑↓ Temporary storage (GiB)	↑↓ Premium disk support
A6	Standard	General purpose	4	28	8	8x500		No
A7	Standard	General purpose	8	56	16	16x500		No
A8_v2 ⓘ	Standard	General purpose	8	16	16	16x500	80	No
A8m_v2 ⓘ	Standard	General purpose	8	64	16	16x500	80	No
B12ms ⓘ	Standard	General purpose	12	48	16	4320	96	Yes
B16ms ⓘ	Standard	General purpose	16	64	32	4320	128	Yes
B1s ⓘ	Standard	General purpose	1	0.5	2	160	4	Yes
B1ms ⓘ	Standard	General purpose	1	2	2	640	4	Yes
B1s ⓘ	Standard	General purpose	1	1	2	320	4	Yes
B20ms ⓘ	Standard	General purpose	20	80	32	4320	160	Yes
B2ms ⓘ	Standard	General purpose	2	8	4	1920	16	Yes
B2s ⓘ	Standard	General purpose	2	4	4	1280	8	Yes
B4ms ⓘ	Standard	General purpose	4	16	8	2880	32	Yes
B8ms ⓘ	Standard	General purpose	8	32	16	4320	64	Yes
D1_v2 ⓘ	Standard	General purpose	1	3.5	4	4x500	50	No
D11_v2 ⓘ	Standard	Memory optimized	2	14	8	8x500	100	No
D12_v2 ⓘ	Standard	Memory optimized	4	28	16	16x500	200	No
D13_v2 ⓘ	Standard	Memory optimized	8	56	32	32x500	400	No
D14_v2 ⓘ	Standard	Memory optimized	16	112	64	64x500	800	No
D15_v2 ⓘ	Standard	Memory optimized	20	140	64	64x500	1000	No
D16_v3 ⓘ	Standard	General purpose	16	64	32	32x500	400	No
D16s_v3 ⓘ	Standard	General purpose	16	64	32	25600	128	Yes

The image above shows just a small set of the series and size possibilities. For each option, you can see the number of Virtual CPUs, the amount of RAM, the number of Data disks, the Max IOPS, the temporary storage provided and whether Premium storage is supported.

For more information about VM size best practices, see [Best practices for SQL Server on Azure VMs](#).

Azure Marketplace

The Azure Marketplace is essentially a centralized location that provides the ability to create Azure resources based on a pre-designed template. For example, you can quickly create a SQL Server 2019 instance on Windows Server 2019 with a couple of clicks of the mouse along with some basic information, such as the virtual machine name as well as some SQL Server configuration information. Once provided, Azure Resource Manager will initiate the creation of the virtual machine and within minutes it will be up and running.

The blade for SQL Server 2019 on Windows Server 2019 in the Azure Marketplace is shown below. This blade gives you the option of pre-set configurations that support OLTP or Data Warehouse workloads and allow you to specify storage, patching, and backup options.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with navigation links: Create a resource, Home, Dashboard, All services, FAVORITES (which is selected), All resources, Resource groups, App Services, SQL servers, SQL managed instances, SQL databases, Azure Cosmos DB, and Virtual machines. The main content area has a breadcrumb trail: Home > New > Marketplace > SQL Server 2019 on Windows Server 2019. The page title is "SQL Server 2019 on Windows Server 2019" by Microsoft. It features a red icon with a monitor and the text "SQL Server 2019 on Windows Server 2019". Below it, there's a "Select a software plan" dropdown set to "Free SQL Server License: SQL 2019 ...", a "Create" button, and a "Start with a pre-set configuration" link. A "Save for later" button is also present. The "Overview" tab is selected, showing "SQL Server 2019 Standard, Enterprise and Developer image on Windows Server 2019". Other tabs include "Plans", "Useful Links", "Documentation", "SQL Server 2019 information", "Support forum", and "Pricing details".

The disadvantage of using the portal to create Azure resources is that it is not an easily repeatable process. However, it is easy to get started with the portal, where you can quickly get up and running resources.

SQL Server configuration

When provisioning SQL Server to an Azure virtual machine, you can also configure specific SQL Server settings such as, **Security and Networking**, **SQL Authentication preferences**, **SQL instance settings**, and a few other options. These options are located on the **SQL Server settings** tab, as shown in the image below.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

[Home](#) > [Azure SQL](#) > [Select SQL deployment option](#) >

Create a virtual machine

[Basics](#) [Disks](#) [Networking](#) [Management](#) [Advanced](#) [SQL Server settings](#) [Tags](#) [Review + create](#)

Security & Networking

SQL connectivity *

Port *

SQL Authentication

SQL Authentication

Azure Key Vault integration

Storage configuration

Customize performance, size, and workload type to optimize storage for this virtual machine. For optimal performance, separate drives will be created for data and log storage by default. [Learn more about SQL Server best performance practices.](#)

Storage

Not available

[Change configuration](#)

SQL instance settings

Customize additional SQL instance settings including collation, MAXDOP, server memory limit and optimize for ad-hoc workload.

Instance settings

Default configuration

MAXDOP: 0

SQL Server memory limits: 0 - 2147483647 MB

Collation: SQL_Latin1_General_CI_AS

[Change SQL instance settings](#)

SQL Server License

Save up to 43% with licenses you already own. Already have a SQL Server license? [Learn more](#)

SQL Server License

No Yes

Automated patching

Set a patching window during which all Windows and SQL patches will be applied.

Automated patching

Enabled

Sunday at 2:00

[Change configuration](#)

Automated backup

Automated backup

R Services(Advanced Analytics)

SQL Server Machine Learning Services
(In-Database) 

[Disable](#) [Enable](#) 

[Review + create](#)

< Previous

Next : Tags >

For more information about the SQL Server settings available when creating a virtual machine, see [Provision SQL Server on Azure VM \(Azure portal\)](#).

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

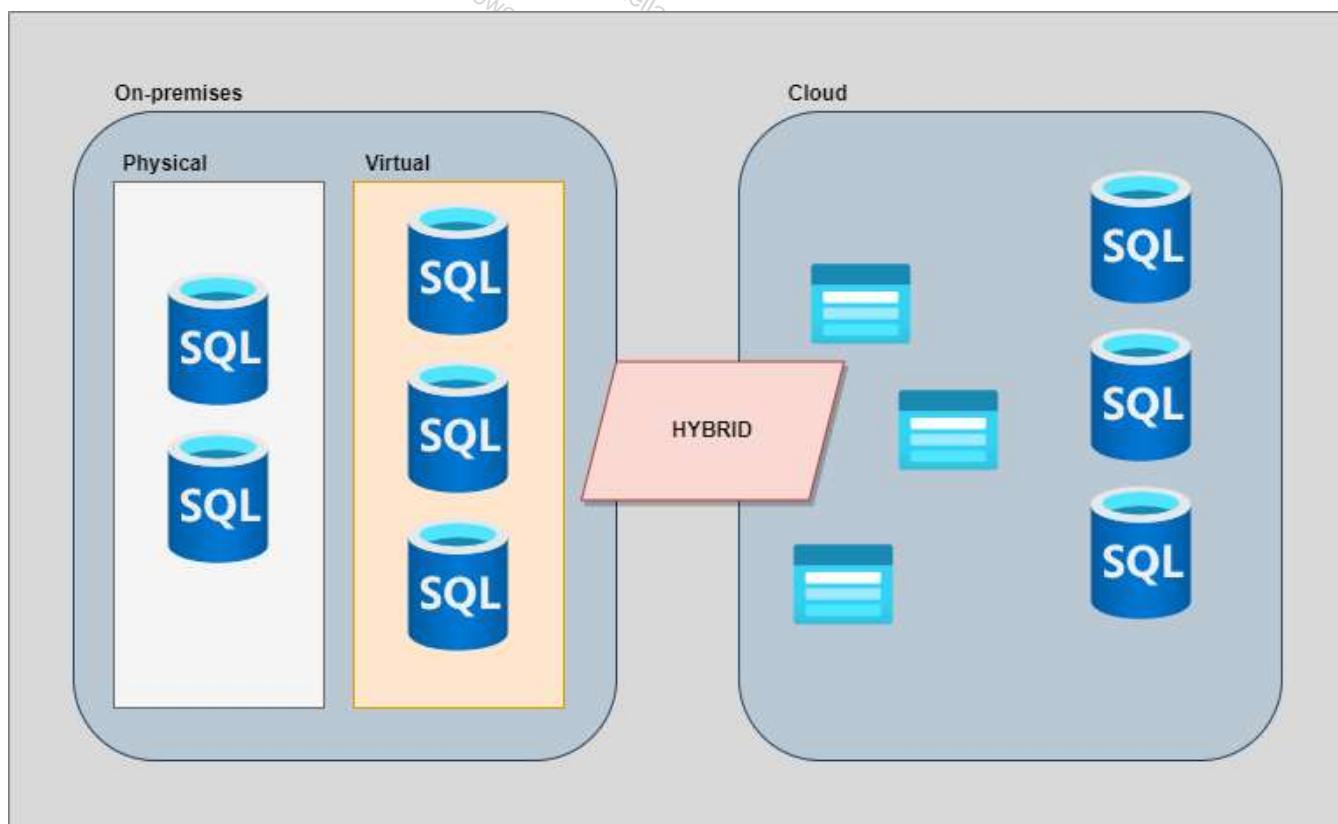
This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Understand hybrid scenarios

Suppose your organization has a significant investment in SQL Server infrastructure on-premises and in local data centers. In that case, it's crucial to be aware that using the cloud isn't an all-or-nothing proposition. There are ways to use existing on-premises infrastructure in a hybrid capacity with Azure to improve operational resiliency and reduce cost.

Implementing a hybrid infrastructure is also an excellent first step in evaluating cloud computing for organizations that have been traditionally on-premises and skeptical of the cloud. It's common for today's organizations to have a mixture of physical and virtualized deployments of SQL Server on-premises, both of which may extend to the cloud as part of a hybrid solution. The hybrid SQL Server platform offers the benefits of both on-premises and cloud services; it's a complimentary middle ground between them. The cloud component typically uses IaaS services such as storage or SQL Server virtual machines, as indicated below.



In addition to extending on-premises solutions, the same patterns may be applied to existing alternative cloud solutions, thus enabling cloud-to-cloud hybrid implementations. Let's review some of the most common hybrid scenarios for SQL Server.

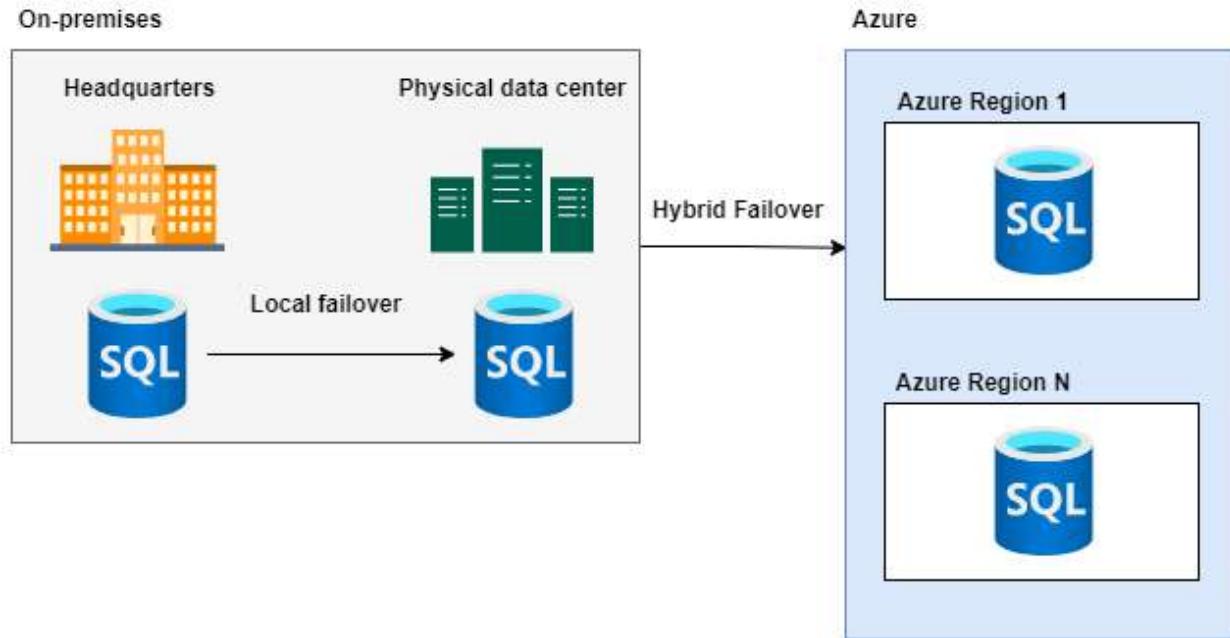
Hybrid scenarios for SQL Server

Let's review a few strategies for when deploying a hybrid solution for SQL Server.

Disaster Recovery

Disaster Recovery is the most common scenario for a hybrid deployment of SQL Server. Disaster recovery means that organizations ensure business continuity in the light of catastrophic events. Organizations may distribute deployments across multiple data centers for failover in an on-premises approach. These data centers typically

reside within the same geographic region as the organization, thus susceptible to more significant regional disasters. Physical data centers are also costly to deploy, monitor, and maintain. Arguably, the cost of spinning up Azure SQL Server virtual machines in various geographical regions is much less than establishing a new physical data center in another geography.



In this hybrid approach, Azure is used for **DR failover** (to one or more regions) while the regular day-to-day processing continues to use on-premises servers for local high availability.

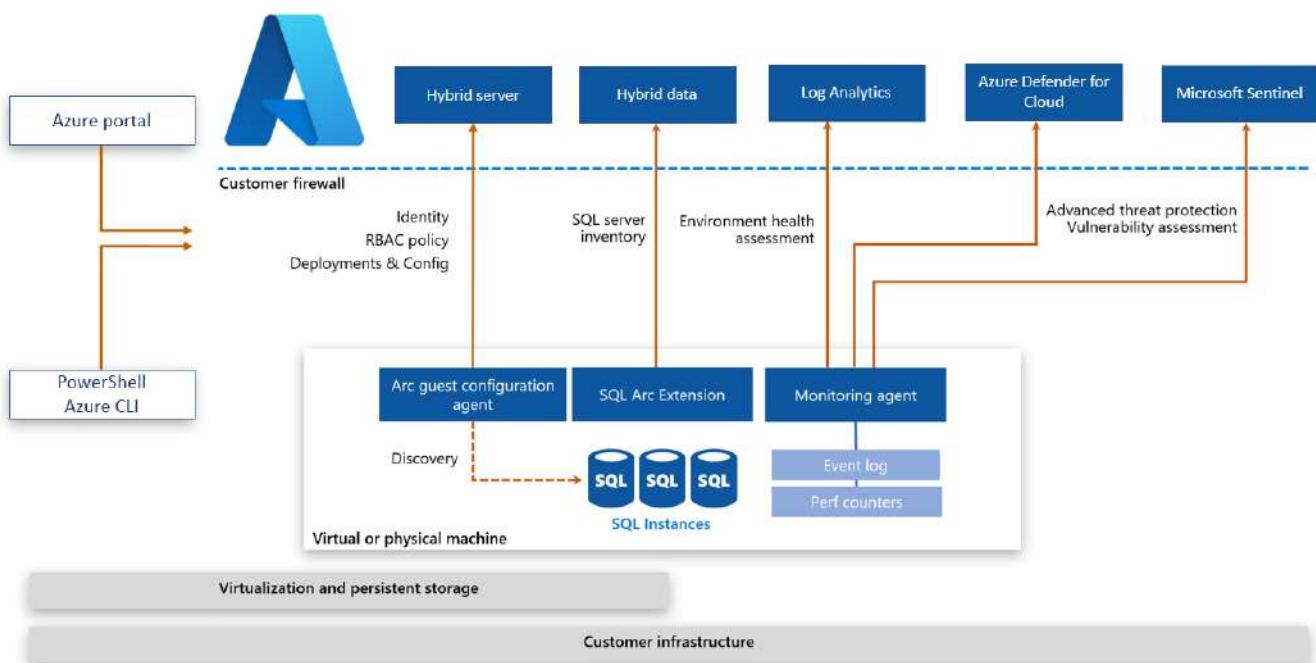
SQL Server Backups

SQL Server Backups is another common hybrid scenario. Backups may be done directly into Azure Storage via URL or Azure file share (SMB). This scenario protects against data loss when on-site backup storage fails. In addition, these backups may also be restored to virtual machines in Azure and tested as part of Disaster Recovery procedures.

Another scenario uses Azure Storage to store on-premises **SQL Server data files for user databases**. Note that these are user files and not system databases. In the case of local storage failure, the user files are safely stored in the cloud, preventing data loss. In addition, by using Azure storage, there are **built-in reliability guarantees** so storing these files in the cloud is more resilient. For this hybrid scenario, it's essential to keep the network communication secure, evaluate the network latency of the solution, and ensure the storage account is locked down using ACLs and Azure Active Directory.

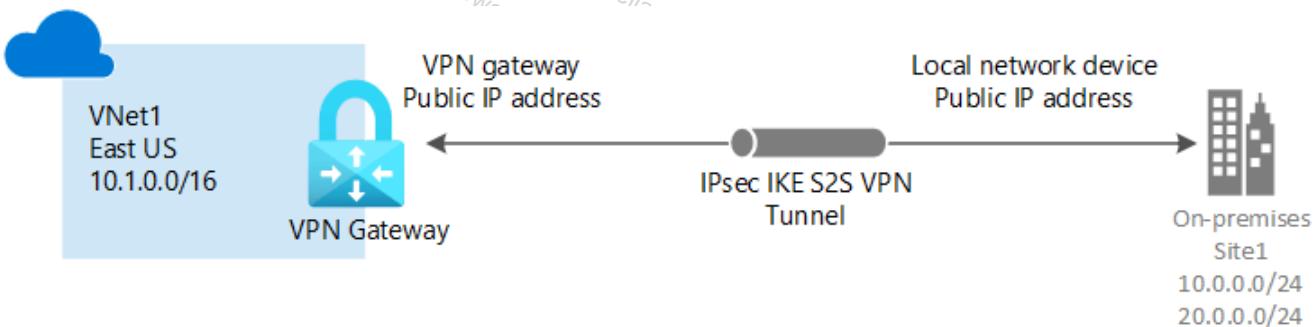
Azure Arc enabled SQL Servers

Having **Azure Arc enabled SQL Servers** extends and centralizes Azure management services to SQL Server instances hosted on-premises, in your data centers, on the edge, and in multi-cloud environments. In this hybrid scenario, Azure Arc enables the inventory of all registered SQL Server deployments and assesses their configurations, usage patterns, and security to provide actions and recommendations based on best practices. By using Azure Arc enabled SQL Servers, you gain the benefits of centralized server management. You also get Azure Defender real-time security alerts and vulnerability reporting on both on-premises SQL Servers and their host operating systems. In addition, Azure Sentinel can provide more security threat introspection if necessary.

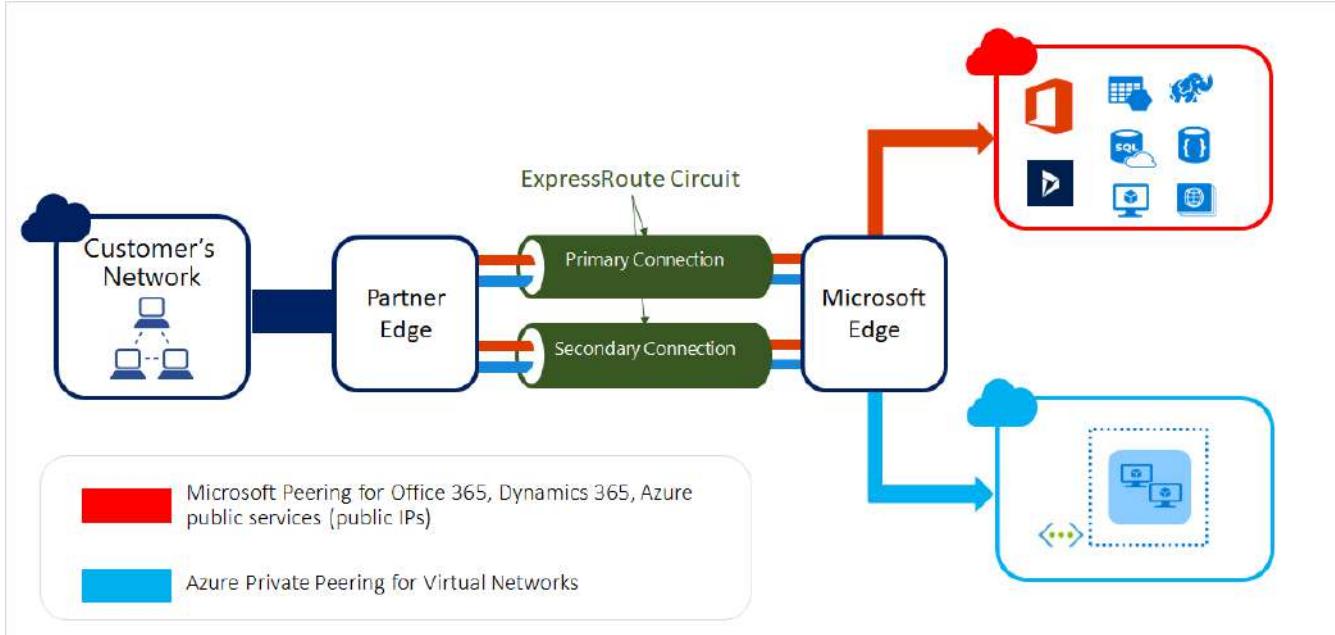


Security considerations

When deploying a hybrid SQL solution, all core infrastructure, such as Active Directory and DNS, must exist on-premises and in Azure. In addition, secure two-way communication must exist between the on-premises network and Azure. This secured communication can take the form of a **site-to-site (S2S) VPN** or a dedicated **ExpressRoute** tunnel. When evaluating different connectivity methods, it's vital to determine the amount of latency acceptable for your organization. Regardless of the solution chosen, network security must also be at the forefront of the implementation.



The image above shows the benefit of an S2S VPN solution is that it tends to cost less, and its implementation is a common task amongst network engineers. However, with this solution, all communication occurs over the public internet and is limited by the organization's internet speeds.



As we can see above, while the ExpressRoute solution tends to be more costly, it also provides the best security and lowest latency as all communication flows over a direct secured channel independent of the public internet. However, common detractors for this solution include overall cost and the inability to apply ExpressRoute between cloud providers in a multi-cloud solution.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Explore performance and security

The Azure ecosystem offers several performance and security options for SQL Server instance on Azure virtual machine. Each option provides several capabilities, such as different disk types that meet the capacity and performance requirements of your workload.

Storage considerations

SQL Server requires good storage performance to deliver robust application performance, whether it be an on-premises instance or installed in an Azure VM. Azure provides a wide variety of storage solutions to meet the needs of your workload. While Azure offers various types of storage (blob, file, queue, table) in most cases SQL Server workloads will use Azure managed disks. The exceptions are that a Failover Cluster Instance can be built on file storage and backups will use blob storage. Azure-managed disks act as a block-level storage device that is presented to your Azure VM. Managed disks offer a number of benefits including 99.999% availability, scalable deployment (you can have up to 50,000 VM disks per subscription per region), and integration with availability sets and zones to offer higher levels of resiliency in case of failure.

Azure-managed disks all offer two types of encryption. Azure Server-side encryption is provided by the storage service and acts as encryption-at-rest provided by the storage service. Azure Disk Encryption uses BitLocker on Windows, and DM-Crypt on Linux to provide OS and Data disk encryption inside of the VM. Both technologies integrate with Azure Key Vault and allow you to bring your own encryption key.

Each VM will have at least two disks associated with it:

- **Operating System disk – Each virtual machine will require an operating system disk that contains the boot volume. This disk would be the C: drive in the case of a Windows platform virtual machine, or /dev/sda1 on Linux. The operating system will be automatically installed on the operating system disk.**
- **Temporary disk – Each virtual machine will include one disk used for temporary storage. This storage is intended to be used for data that does not need to be durable, such as page files or swap files. Because the disk is temporary, you should not use it for storing any critical information like database or transaction log files as they will be lost during maintenance or a reboot of the virtual machine. This drive will be mounted as D:\ on Windows, and /dev/sdb1 on Linux.**

Additionally, you can and should add additional data disks to your Azure VMs running SQL Server.

- **Data disks – The term data disk is used in the Azure portal, but in practice these are just additional managed disks added to a VM. These disks can be pooled to increase the available IOPs and storage capacity, using Storage Spaces on Windows or Logical Volume Management on Linux.**

Furthermore, each disk can be one of several types:

Feature	Ultra Disk	Premium SSD	Standard SSD	Standard HDD
Disk type	SSD	SSD	SSD	HDD
Best for	IO-intensive workload	Performance sensitive workload	Lightweight workloads	Backups, non-critical workloads
Max disk size	65,536 GiB	32,767 GiB	32,767 GiB	32,767 GiB

Feature	Ultra Disk	Premium SSD	Standard SSD	Standard HDD
Max throughput	2,000 MB/s	900 MB/s	750 MB/s	500 MB/s
Max IOPS	160,000	20,000	6,000	2,000

The best practices for SQL Server on Azure recommend using Premium Disks pooled for increased IOPs and storage capacity. Data files should be stored in their own pool with read-caching on the Azure disks.

Transaction log files will not benefit from this caching, so those files should go into their own pool without caching. TempDB can optionally go into its own pool, or using the VM's temporary disk, which offers low latency since it is physically attached to the physical server where the VMs are running. Properly configured Premium SSD will see latency in single digit milliseconds. For mission critical workloads that require latency lower than that, you should consider Ultra SSD.

Security considerations

There are several industry regulations and standards that Azure complies with that makes it possible to build a compliant solution with SQL Server running in a virtual machine.

Microsoft Defender for SQL

Microsoft Defender for SQL provides Azure Security Center security features such as vulnerability assessments and security alerts.

Azure Defender for SQL can be used to identify and mitigate potential vulnerabilities in your SQL Server instance and database. The vulnerability assessment feature can detect potential risks in your SQL Server environment and help you remediate them. It also provides insight into your security state and actionable steps to resolve security issues.

Azure Security Center

Azure Security Center is a **unified security management system that evaluates and offers opportunities for improving several security aspects of your data environment**. Azure Security Center provides a **comprehensive view of the security health of all your hybrid cloud assets**.

Performance considerations

Most of the existing on-premises SQL Server performance features are also available on Azure virtual machines (VMs). **Among the options offered is data compression, which can improve the performance of I/O-intensive workloads while decreasing the size of the database. Similarly, table and index partitioning can improve query performance of large tables**, while improving performance and scalability.

Table partitioning

Table partitioning provides many benefits, but often times this strategy is only considered when the table becomes large enough that starts compromising query performance. Identifying which tables are candidates for table partitioning is a good practice that could lead to less disruptions and interventions. **When you filter your data using your partition column, only a subset of the data is accessed, not the entire table. Similarly, maintenance operations on a partitioned table will reduce maintenance duration**, for example, by compressing specific data in a particular partition or rebuilding specific partitions of an index.

There are four main steps required when defining a table partition:

- The filegroups creation, which defines the files involved when the partitions are created.
- The partition function creation, which defines the partition rules based on the specified column.
- The partition scheme creation, which defines the filegroup of each partition.
- The table to be partitioned.

The example below illustrates how to create a partition function for January 1, 2021 through December 1, 2021, and distribute the partitions across different filegroups.

```
-- Partition function
CREATE PARTITION FUNCTION PartitionByMonth (datetime2)
    AS RANGE RIGHT
    -- The boundary values defined is the first day of each month, where the table will be
    partitioned into 13 partitions
    FOR VALUES ('20210101', '20210201', '20210301',
                '20210401', '20210501', '20210601', '20210701',
                '20210801', '20210901', '20211001', '20211101',
                '20211201');

-- The partition scheme below will use the partition function created above, and assign each
partition to a specific filegroup.
CREATE PARTITION SCHEME PartitionByMonthSch
    AS PARTITION PartitionByMonth
    TO (FILEGROUP1, FILEGROUP2, FILEGROUP3, FILEGROUP4,
        FILEGROUP5, FILEGROUP6, FILEGROUP7, FILEGROUP8,
        FILEGROUP9, FILEGROUP10, FILEGROUP11, FILEGROUP12);

-- Creates a partitioned table called Order that applies PartitionByMonthsch partition
scheme to partition the OrderDate column
CREATE TABLE Order ([Id] int PRIMARY KEY, OrderDate datetime2)
    ON PartitionByMonthsch (OrderDate) ;
GO
```

Data compression

SQL Server offers different options for compressing data. While SQL Server still stores compressed data on 8 KB pages, when the data is compressed, more rows of data can be stored on a given page, which allows the query to read fewer pages. Reading fewer pages has a twofold benefit: it reduces the amount of physical IO performed and it allows more rows to be stored in the buffer pool, making more efficient use of memory. We recommend enabling database page compression where appropriate.

The tradeoffs to compression are that it does require a small amount of CPU overhead, however, in most cases the storage IO benefits far outweigh any additional processor usage.

The screenshot shows a SQL Server Management Studio window with two queries. The first query is against the `Production.TransactionHistory` table, which is uncompressed. The second query is against the `Production.TransactionHistory_Page` table, which is page-compressed. Both queries count rows from January 1, 2008, to the present.

```

SELECT COUNT(*) FROM Production.TransactionHistory WHERE TransactionDate > '2008-01-01'

SELECT COUNT(*) FROM Production.TransactionHistory_Page WHERE TransactionDate > '2008-01-01'

```

The results show that the compressed table requires significantly fewer logical reads:

```

(1 row affected)
Table 'TransactionHistory'. Scan count 1, logical reads 992, physical reads 0

(1 row affected)
Table 'TransactionHistory_Page'. Scan count 1, logical reads 273, physical reads 0

Completion time: 2020-04-22T14:55:13.3744311+00:00

```

The image above shows this performance benefit. These tables have same underlying indexes; the only difference is that the clustered and nonclustered indexes on the `Production.TransactionHistory_Page` table are page compressed. The query against the page compressed object performs 72% fewer logical reads than the query that uses the uncompressed objects.

Compression is implemented in SQL Server at the object level. Each index or table can be compressed individually, and you have the option of compressing partitions within a partitioned table or index. You can evaluate how much space you will save by using the `sp_estimate_data_compression_savings` system stored procedure. Prior to SQL Server 2019, this procedure did not support columnstore indexes, or columnstore archival compression.

- **Row compression** - Row compression is fairly basic and does not incur much overhead; however, it does not offer the same amount of compression (measured by the percentage reduction in storage space required) that page compression may offer. Row compression basically stores each value in each column in a row in the minimum amount of space needed to store that value. It uses a variable-length storage format for numeric data types like integer, float, and decimal, and it stores fixed-length character strings using variable length format.
- **Page compression** - Page compression is a superset of row compression, as all pages will initially be row compressed prior to applying the page compression. Then a combination of techniques called prefix and dictionary compression are applied to the data. Prefix compression eliminates redundant data in a single column, storing pointers back to the page header. After that step, dictionary compression searches for repeated values on a page and replaces them with pointers, further reducing storage. The more redundancy in your data, the greater the space savings when you compress your data.

- **Columnstore archival compression** - Columnstore objects are always compressed, however, they can be further compressed using archival compression, which uses the Microsoft XPRESS compression algorithm on the data. This type of compression is best used for data that is infrequently read, but needs to be retained for regulatory or business reasons. While this data is further compressed, the CPU cost of decompression tends to outweigh any performance gains from IO reduction.

Additional options

Below is a list of additional SQL Server features and actions to consider for production workloads:

- Enable backup compression
- Enable instant file initialization for data files
- Limit autogrowth of the database
- Disable autoshrink/autoclose for the databases
- Move all databases to data disks, including system databases
- Move SQL Server error log and trace file directories to data disks
- Set max SQL Server memory limit
- Enable lock pages in memory
- Enable optimize for adhoc workloads for OLTP heavy environments
- Enable Query Store.
- Schedule SQL Server Agent jobs to run DBCC CHECKDB, index reorganize, index rebuild, and update statistics jobs
- Monitor and manage the health and size of the transaction log files

For more information about performance best practices, see [Best practices for SQL Server on Azure VMs](#).

Explain high availability and disaster recovery options

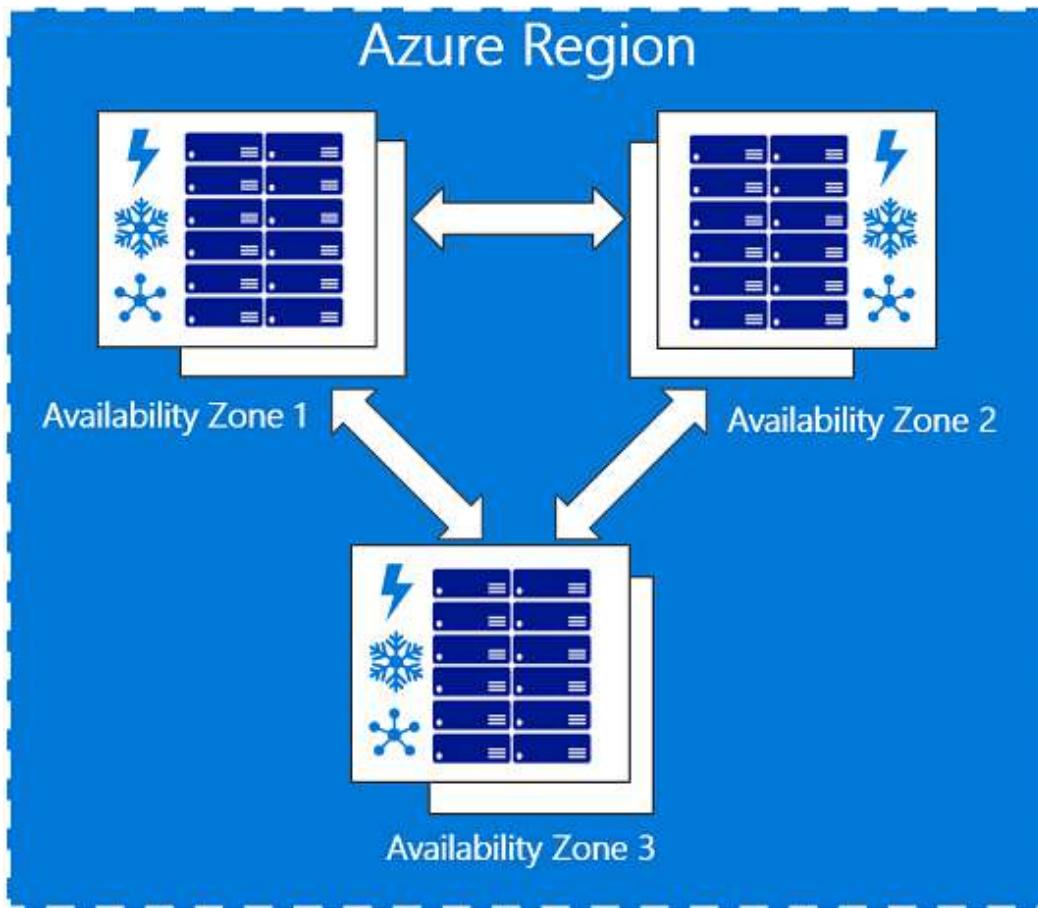
Beyond its built-in high availability, the Azure platform offers two options for providing higher levels of availability for VM and some PaaS workloads. Availability Zones and Availability Sets protect your workloads from planned maintenance activity and potential hardware failures.

High availability options

Most SQL Server high availability solutions are available on Azure virtual machines (VMs). In an Azure-only solution, the entire HADR system runs in Azure. In a hybrid configuration, part of the solution runs in Azure and the other part runs on-premises in your organization. The flexibility of the Azure environment enables you to move partially or completely to Azure to satisfy the budget and HADR requirements of your SQL Server database systems.

Availability Zones

Availability Zones are unique physical locations within a region. Each zone is made up of one or more data centers equipped with independent power, cooling, and networking. Within Azure regions that support Availability Zones, you can specify in which zone you want the virtual machine to reside when you choose to use Available Zones during VM creation. There are three Availability Zones within each supported Azure region. Availability Zones provide high availability against data center failures when you deploy multiple VMs into different zones. In addition, they also provide a means for Microsoft to perform maintenance (using a grouping called an update domain) within each region by only updating one zone at any given time. You can spread out your virtual machine ecosystem across three zones in the region. Utilizing Availability Zones in conjunction with your Azure virtual machines raises your uptime to four nines (99.99%) which equates to a maximum of 52.60 minutes of downtime per year. You can identify which Azure regions support Availability Zones at docs.microsoft.com. If Availability Zones are available in your region, and your application can support the minimal cross-zone latency, Availability Zones will provide the highest level of availability for your application.



In the image above, you can see the availability zone configuration. When you deploy a VM into a region with an availability zone you will be presented with the option to deploy in Zone 1, 2, and 3. These zones are logical representations of physical data centers, which means a deployment to Zone 1 in one subscription, does not mean that Zone 1 represents the same data center in another subscription.

Availability Sets

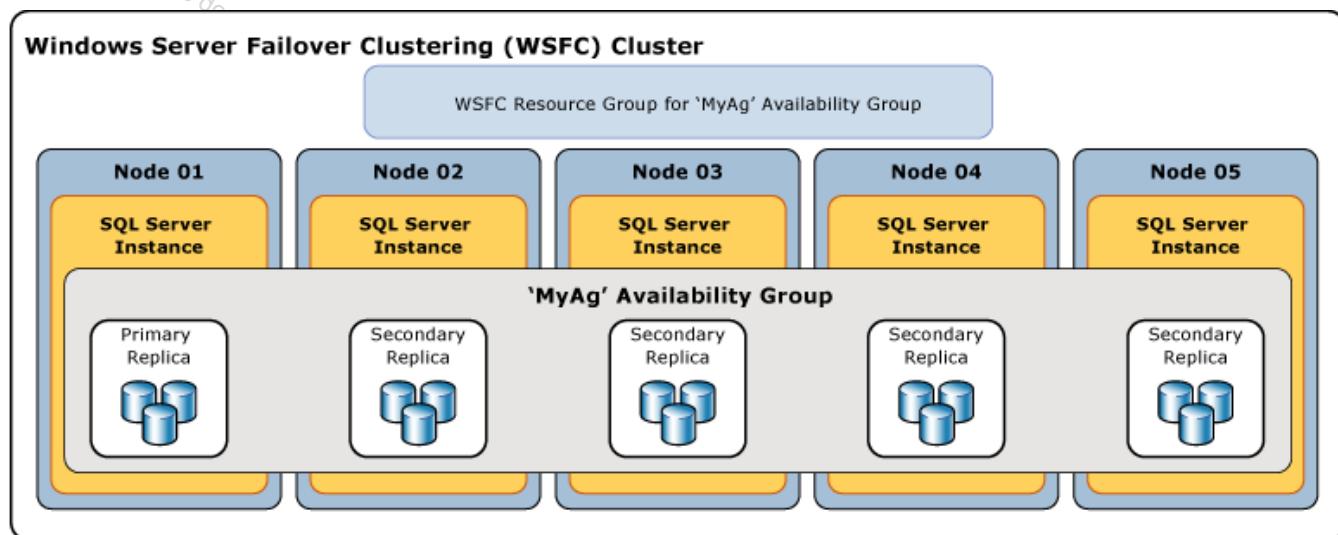
Availability sets are similar to **Availability Zones**, except instead of spreading workloads across data centers in a region, they spread workloads across servers and racks in a data center. Since nearly all workloads in Azure are virtual, you can use availability sets in order to guarantee that the two VMs containing your Always On Availability Group members are not running on the same physical host. Availability sets can provide up to 99.95% availability, and should be used when Availability Zones are unavailable in a region, or an application cannot tolerate intra-zone latency.

Always On availability groups (AG)

Always On availability groups can be implemented between two or more (up to a maximum of nine) SQL Server instances running on Azure virtual machines or across an on-premises data center and Azure. In an availability group, database transactions are committed to the primary replica, and then the transactions are sent either synchronously or asynchronously to all secondary replicas. The physical distance between the servers (that is, whether or not they are in the same Azure region) dictates which availability mode you should choose. Generally, if the workload requires the lowest possible latency or the secondary replicas are geographically spread apart, asynchronous availability mode is recommended. If the replicas are within the same Azure region and the applications can withstand some level of latency, synchronous commit mode should be considered. Synchronous mode will help to ensure that each transaction is committed to one or more secondaries before allowing the application to continue. Always On availability groups provide both high availability and disaster recovery, because

a single availability group can support both synchronous and asynchronous availability modes. The unit of failover for an availability group is a group of databases, and not the entire instance.

Always On Availability Groups can also be used for disaster recovery purposes. **You can implement up to nine replicas of a database across Azure regions, and stretch this architecture even further using Distributed Availability Groups.** Availability Groups ensure that a viable copy of your database(s) is in another location beyond the primary region. By doing so, you help to ensure that your data ecosystem is protected against natural disasters as well as some human made ones.



The image above shows a logical diagram of an Always On Availability Group, running on a Windows Server Failover Cluster. There are one primary and four secondary replicas. In this scenario all five replicas could be synchronous, or some combination of synchronous and asynchronous replicas. Keep in mind that the unit of failover is the group of databases and not the instance. While a failover cluster instance provides HA at an instance level, it doesn't provide disaster recovery.

SQL Server Failover Cluster instances

If you need to protect the entire instance, you could use a SQL Server Failover Cluster Instance (FCI), which provides high availability for an entire instance, in a single region. A FCI doesn't provide disaster recovery without being combined with another feature like availability groups or log shipping. FCIs also require shared storage that can be provided on Azure by using shared file storage or using Storage Spaces Direct on Windows Server.

For Azure workloads, availability groups are the preferred solution for newer deployments, because the shared storage requirement of FCIs increases the complexity of deployments. However, for migrations from on-premises solutions, an FCI may be required for application support.

Disaster Recovery options

While the Azure platform offers 99.9% up time by default, disasters can still occur and affect application uptime. It's important that you have a proper disaster recovery plan in place when you are performing any type of migration. Azure offers us several methods to ensure that your SQL Server on a virtual machine is protected in case of a disaster. There are two components to this protection. First, there are Azure platform options like geo-replicated storage for backups and Azure Site Recovery, which is an all-encompassing disaster recovery solution for all of your workloads. Second, there are SQL Server specific offerings like Availability Groups and backups.

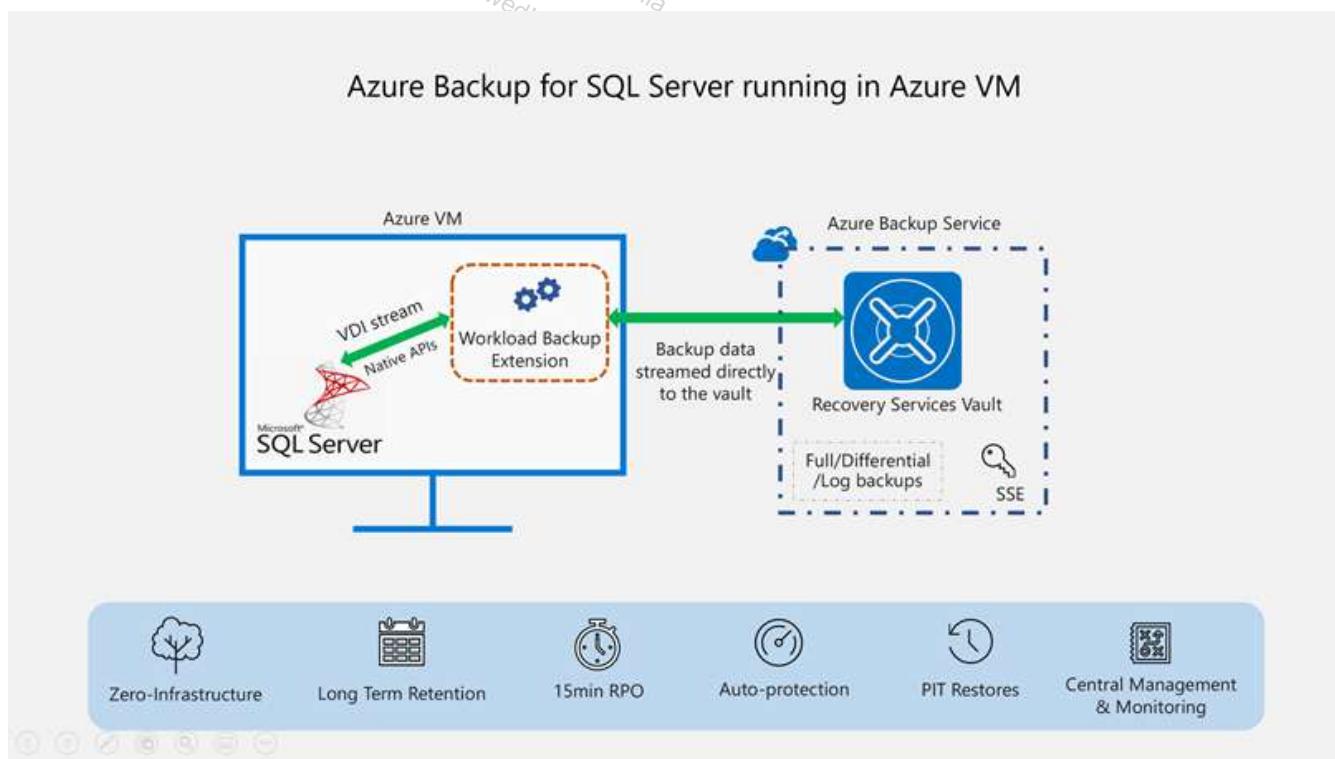
Native SQL Server backups

Backups are considered the life blood of any database administrator and it's not any different when working with a cloud solution. With SQL Server on an Azure virtual machine, you have granular control of when backups occur and where they're stored. You can use SQL agent jobs to back up directly to a URL linked to Azure blob storage. Azure provides the option to use geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS) to ensure that your backup files are stored safely across the geographic landscape.

Additionally as part of the Azure SQL VM service provider, you can have your backups automatically managed by the platform.

Azure Backup for SQL Server

The Azure Backup solution requires an agent to be installed on the virtual machine. The agent then communicates with an Azure service that manages automatic backups of your SQL Server databases. Azure Backup also provides a central location that you can use to manage and monitor the backups to ensure meeting any specified RPO/RTO metrics.

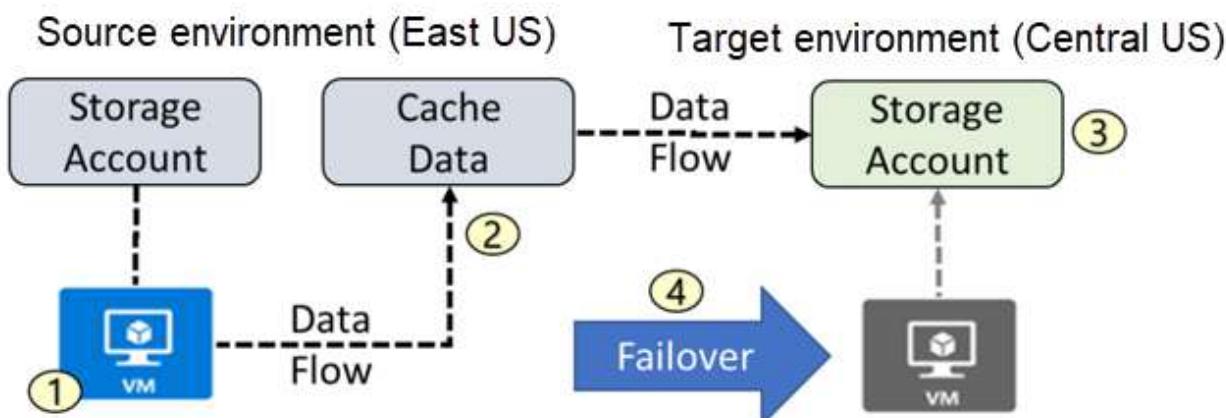


As shown above, the Azure Backup solution is a comprehensive, enterprise backup solution that provides long-term data retention, automated management, and additional data protection. This option costs more than simply performing your own backups, or using the Azure resource provider for SQL Server, but does offer a more complete backup feature set.

Azure Site Recovery

Azure Site Recovery is a low-cost solution that will perform block level replication of your Azure virtual machine. This service offers various options, including the ability to test and verify your disaster recovery strategy. This solution is best used for stateless environments (for example, web servers) versus transactional database virtual machines.

Azure Site Recovery is supported for use with SQL Server, but keep in mind that you will need to set a higher recovery point which means potential loss. In this case, your RTO will essentially be your RPO.



1. VM is registered with Azure Site Recovery
 2. Data is continuously replicated to cache
 3. Cache is replicated to the target storage account
 4. During failover the virtual machine is added to the target environment

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What type of storage offers the lowest latency in Azure?

- Ultra SSD
- Premium SSD
- Standard SSD

Check Answers

Multiple choice

To reduce the cost of an Azure SQL Server VM you intend to run full time for three years, which option should you choose?

- Availability set
- Azure Reserved VM Instances
- Pay as You Go Licensing

Check Answers

Multiple choice

Which option should you choose to spread workloads across data centers in a region?

- Availability sets
- Availability zones
- Availability units

Check Answers

Multiple choice

Which service performs block level replication of your Azure virtual machine?

- Azure Site Recovery
- Azure Backup for SQL Server
- Availability Groups

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

Azure provides a great deal of flexibility for deploying SQL Server to an Azure virtual machine. The hybrid licensing options reduce your cost. It also allows for additional protection for high availability and disaster recovery as part of your software assurance benefits. Azure Resource Manager offers many ways to deploy your resources in both a programmatic and graphical fashion.

Now that you've reviewed this module, you should be able to:

- Explore the basics of SQL Server in an Infrastructure as a Service (IaaS) offering
- Learn how hybrid scenario works
- Explore performance and security options available
- Understand the high availability and disaster recovery options

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Introduction

The Platform as a Service (PaaS) offering for SQL Server can be a great solution for certain workloads. The PaaS offering provides less granular control over the infrastructure. It also relegates management of the underlying components (memory, CPU, storage, operating system, etc.) to Microsoft Azure.

This module will focus on ways to provision and deploy Azure SQL Database, Azure SQL managed instances and Azure SQL Edge, as well as provide guidance on the various options when performing a migration to these platforms.

Learning objectives

At the end of this module, you'll be able to:

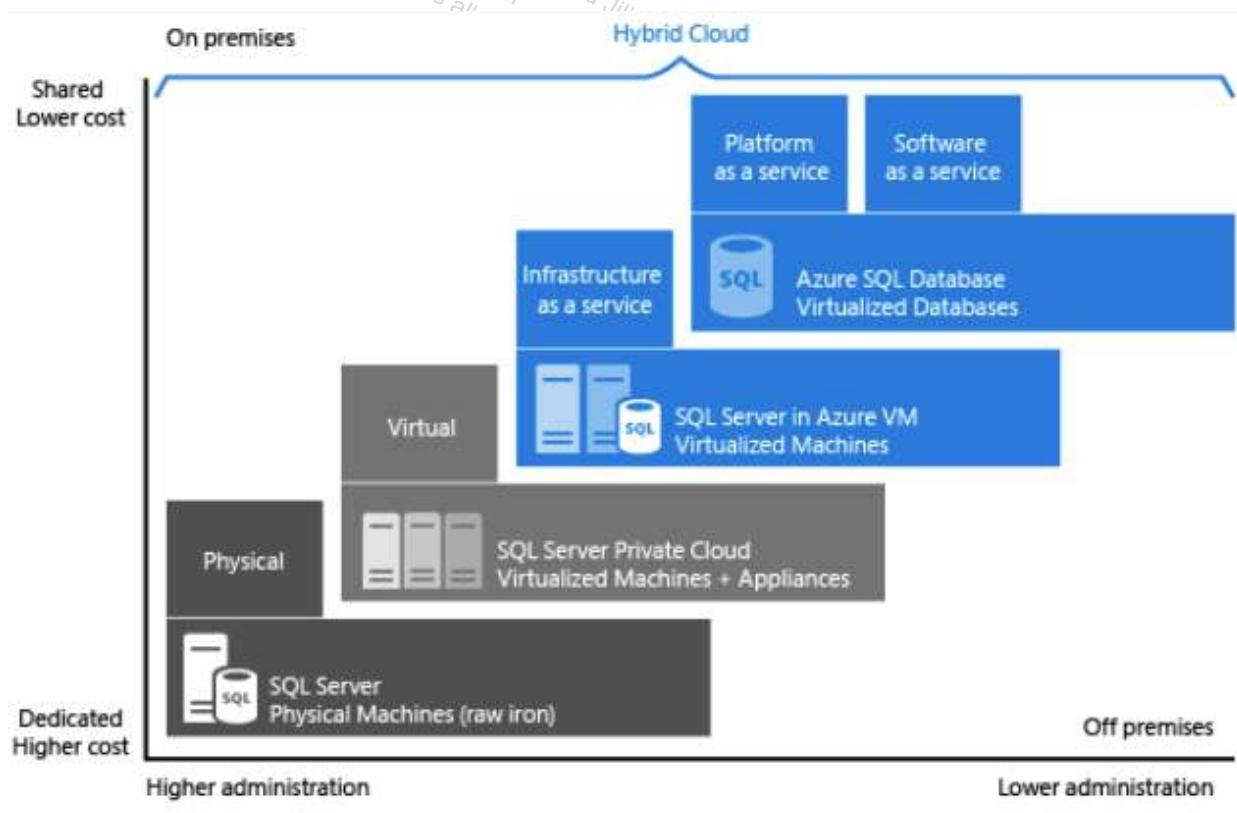
- Understand PaaS provisioning and deployment options
- Understand elastic pools and hyperscale features
- Examine SQL Managed Instances
- Understand SQL Edge

Explain PaaS options for deploying SQL Server in Azure

Platform as a Service (PaaS) provides a complete development and deployment environment in the cloud, which can be used for simple cloud-based applications as well as for advanced enterprise applications.

Azure SQL Database and Azure SQL Managed Instance are part of PaaS offering for Azure SQL.

- **Azure SQL Database** – Part of a family of products built upon the SQL Server engine, in the cloud. It gives developers a great deal of flexibility in building new application services, and granular deployment options at scale. SQL Database offers a low maintenance solution that can be a great option for certain workloads.
- **Azure SQL Managed Instance** – It is best for most migration scenarios to the cloud as it provides fully managed services and capabilities.



As seen in the image above, each offering provides a certain level of administration you have over the infrastructure, by the degree of cost efficiency.

Deployment models

Azure SQL Database is available in two different deployment models:

- **Single database** – a single database that is billed and managed on a per database level. You manage each of your databases individually from scale and data size perspectives. Each database deployed in this model has its own dedicated resources, even if deployed to the same logical server.
- **Elastic Pools** – a group of databases that are managed together and share a common set of resources. Elastic pools provide a cost-effective solution for software as a service application model, since resources are

shared between all databases. You can configure resources based either on the DTU-based purchasing model or the vCore-based purchasing model.

Purchasing model

In Azure, all services are backed by physical hardware, and you can choose from two different purchasing models:

Database Transaction Unit (DTU)

DTUs are calculated based on a formula combining compute, storage, and I/O resources. It is a good choice for customers who want simple, preconfigured resource options.

The DTU purchasing model comes in several different service tiers, such as Basic, Standard and Premium. Each tier has varying capabilities, which provide a wide range of options when choosing this platform.

In terms of performance, the Basic tier is used for less demanding workloads, while Premium is used for intensive workload requirements.

Compute and storage resources are dependent on the DTU level, and they provide a range of performance capabilities at a fixed storage limit, backup retention, and cost.

NOTE: DTU purchasing model is only supported by Azure SQL Database.

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

vCore

The vCore model allows you to purchase a specified number of vCores based on your given workloads. vCore is the default purchasing model when purchasing Azure SQL Database resources. vCore databases have a specific relationship between the number of cores and the amount of memory and storage provided to the database. vCore purchasing model is supported by either Azure SQL Database and Azure SQL Managed Instance.

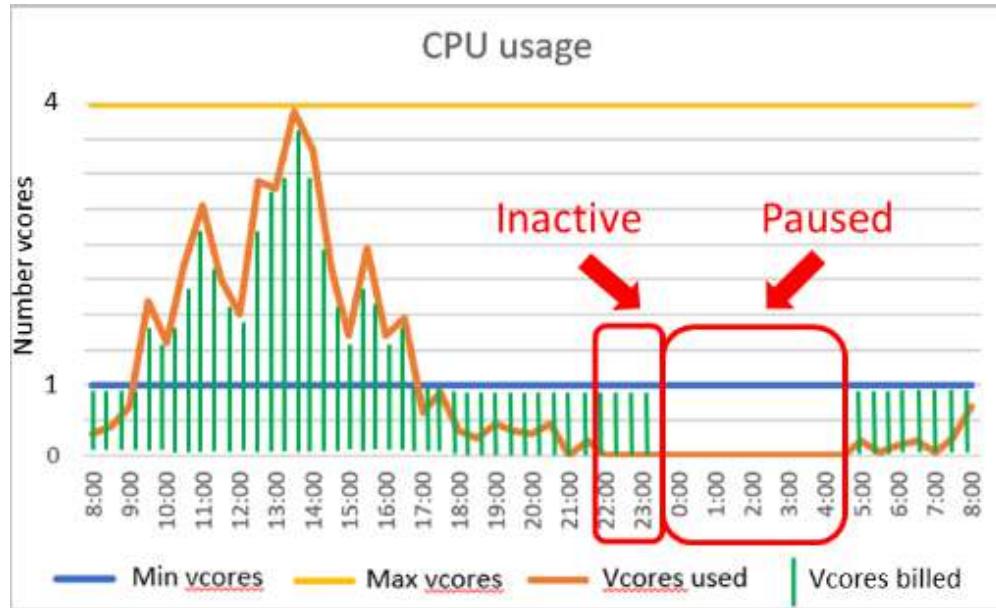
You can purchase vCore databases in three different service tiers as well:

- **General Purpose** – This tier is for general purpose workloads. It is backed by Azure premium storage. It will have higher latency than Business Critical. It also provides the following compute tiers:
 - **Provisioned** – Compute resources are pre-allocated. Billed per hour based on vCores configured.
 - **Serverless** – Compute resources are auto-scaled. Billed per second based on vCores used.
- **Business Critical** – This tier is for high performing workloads offering the lowest latency of either service tier. This tier is backed by local SSDs instead of Azure blob storage. It also offers the highest resilience to failure as well as providing a built-in read-only database replica that can be used to off-load reporting workloads.
- **Hyperscale** – Hyperscale databases can scale far beyond the 4 TB limit the other Azure SQL Database offerings and have a unique architecture that supports databases of up to 100 TB.

Serverless

The name “Serverless” can be a bit confusing as you still deploy your Azure SQL Database to a logical server, to which you connect. Azure SQL Database serverless is a compute tier that will automatically scale up or down the resources for a given database based on workload demand. If the workload no longer requires compute resources,

the database will become “paused” and only storage is billed during the period the database is inactive. When a connection attempt is made, the database will “resume” and become available.



The setting to control pausing is referred to as the autopause delay and has a minimum value of 60 minutes and a maximum value of seven days. If the database has been idle for that period of time, it will then pause.

Once the database has been inactive for the specified amount of time, it will be paused until a subsequent connection is attempted. Configuring a compute autoscaling range and an auto-pause delay affect database performance and compute costs.

Any applications using serverless should be configured to handle connection errors and include retry logic, as connecting to a paused database will generate a connection error.

Another difference between serverless and the normal vCore model of Azure SQL Database is that with serverless you can specify a minimum and maximum number of vCores. Memory and I/O limits are proportional to the range that is specified.

Compute Hardware

Click “Change configuration” to see details for all hardware generations available including memory optimized and compute optimized options

Hardware Configuration
Gen5
up to 16 vCores, up to 48 GB memory
[Change configuration](#)

Max vCores
1 2 4 5 6 8 10 12 14 16 1 vCore

Min vCores
0.5 0.75 1 0.5 vCores

2.02 GB MIN MEMORY 3 GB MAX MEMORY

The image above shows the configuration screen for a serverless database in the Azure portal. You have the option to select a minimum as low as half of a vCore and a maximum as high as 16 vCores.

Serverless is not fully compatible with all the features in Azure SQL Database since some of them require background processes to run at all times, such as:

- Geo-replication
- Long-term backup retention
- A job database in elastic jobs
- The sync database in SQL Data Sync (Data Sync is a service that replicates data between a group of databases)

NOTE: SQL Database serverless is currently only supported in the General Purpose tier in the vCore purchasing model.

Backups

One of the most important features of the Platform as a Service offering is backups. In this case, backups are performed automatically without any intervention from you. Backups are stored in Azure blob geo-redundant storage and by default are retained for between 7 and 35 days, based on the service tier of the database. Basic and vCore databases default to seven days of retention, and on the vCore databases this value can be adjusted by the administrator. The retention time can be extended by configuring long-term retention (LTR), which would allow you to retain backups for up to 10 years.

In order to provide redundancy, you are also able to use read-accessible geo-redundant blob storage. This storage would replicate your database backups to a secondary region of your preference. It would also allow you to read from that secondary region if needed. **Manual backups of databases are not supported, and the platform will deny any request to do so.**

Database Backups are taken on a given schedule:

- **Full** – Once a week
- **Differential** – Every 12 hours
- **Log** – Every 5-10 minutes depending on transaction log activity

This backup schedule should meet the needs of most recovery point/time objectives (RPO/RTO), however, each customer should evaluate whether they meet your business requirements.

There are several options available for restoring a database. Due to the nature of Platform as a Service, you cannot manually restore a database using conventional methods, such as issuing the T-SQL command **RESTORE DATABASE**.

Regardless of which restore method is implemented, it is not possible to restore over an existing database. If a database needs to be restored, the existing database must be dropped or renamed prior to initiating the restore process. Furthermore, keep in mind that depending on the platform service tier, restore times are not guaranteed and could fluctuate. It is recommended that you test the restore process to obtain a baseline metric on how long a restore could potentially take.

The available restore options are:

- **Restore using the Azure portal** – Using the Azure portal you have the option of restoring a database to the same Azure SQL Database server, or you can use the restore to create a new database on a new server in any Azure region.

- **Restore using scripting Languages –** Both PowerShell and Azure CLI can be utilized in order to restore a database.

NOTE: Copy-only backup to Azure blob storage is available for SQL Managed Instance. SQL Database does not support this feature.

For more information about automated backups, see [Automated backups - Azure SQL Database & Azure SQL Managed Instance](#).

Active geo-replication

Geo-replication is a business continuity feature that asynchronously replicates a database to up to four secondary replicas. As transactions are committed to the primary (and its replicas within the same region), the transactions are sent to the secondaries to be replayed. Because this communication is done asynchronously, the calling application does not have to wait for the secondary replica to commit the transaction prior to SQL Server returning control to the caller.

The secondary databases are readable and can be used to offload read-only workloads, thus freeing up resources for transactional workloads on the primary or placing data closer to your end users. Furthermore, the secondary databases can be in the same region as the primary or in another Azure region.

With geo-replication you can initiate a failover either manually by the user or from the application. If a failover occurs, you potentially will need to update the application connection strings to reflect the new endpoint of what is now the primary database.

Failover groups

Failover groups are built on top of the technology used in geo-replication, but provide a single endpoint for connection. The major reason for using failover groups is that the technology provides endpoints, which can be utilized to route traffic to the appropriate replica. Your application can then connect after a failover without connection string changes.

Explore single SQL database

We'll look at several methods for deploying a singleton Azure SQL Database.

Deploying via the portal

The process to create a singleton database through the Azure portal is straightforward. While in the portal, on the left-hand navigation menu, select "SQL Databases". In the resulting slide out dialog, click "Create":

The screenshot shows the Microsoft Azure portal interface. On the left, there's a vertical navigation bar with various service icons: Create a resource, Home, Dashboard, All services, FAVORITES, All resources, Resource groups, App Services, SQL servers, SQL managed instances, SQL databases (which has a red arrow pointing to it), and Azure Cosmos DB. The 'SQL databases' item is highlighted. To the right, a 'New' blade is open under the 'Home > New' path. It features a search bar 'Search the Marketplace'. Below it are sections for 'Azure Marketplace' (with links to Get started, Recently created, and AI + Machine Learning) and 'Popular' services (Windows and IoT). A large section for 'SQL databases' is shown, featuring its icon, the text 'SQL databases', and two buttons: '+ Create' (which has a red arrow pointing to it) and 'View'.

In the blade in the image below, you'll notice that the subscription should already be provided for you. You will need to supply the following information:

- **Resource Group** – If there is an existing resource group that you wish to use, you may select it from the drop-down list. You can click on the “Create new” option if you wish to create a new resource group for this Azure SQL Database.
- **Database Name** – You must provide a database name.
- **Server** – Each database must reside on a logical server. If you have one already in existence in the appropriate region, you may choose to use it. Otherwise, you can click on the **Create new** link and follow the prompts to create a new logical server to host the database.
- **Want to use SQL elastic pool?** – Determine whether to use an elastic pool.

- **Compute + storage** – Determine the appropriate compute resources needed. By default, it will be a Gen5, 2vCore, with 32 GB of storage until something else is selected. Click on **Configure database** to view alternate configuration options.

Create SQL Database

Microsoft

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Dev-Test-Lab

Resource group * ⓘ Contoso

[Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * Widgets

Server ⓘ contososql1 (West US)

[Create new](#)

Want to use SQL elastic pool? * ⓘ Yes No

Compute + storage * ⓘ

General Purpose
Gen5, 2 vCores, 32 GB storage

[Configure database](#)

[Review + create](#) [Next : Networking >](#)

The image below shows the portal blade in which you can configure the database options. Here you will notice that the service tier is General Purpose and compute tier is Provisioned. Provisioned implies that the compute resources are pre-allocated and billed per hour based on the number of configured vCores. The other option is Serverless, which was discussed previously. Serverless is billed per second based on the number of vCores in use.

General Purpose
Scalable compute and storage options
500 - 20,000 IOPS
2-10 ms latency

Hyperscale
On-demand scalable storage
500 - 204,800 IOPS
1-10 ms latency

Business Critical
High transaction rate and high resiliency
5,000 - 204,800 IOPS
1-2 ms latency

Compute tier

- Provisioned** (Selected)
Compute resources are pre-allocated
Billed per hour based on vCores configured
- Serverless**
Compute resources are auto-scaled
Billed per second based on vCores used

Compute Hardware
Click "Change configuration" to see details for all hardware generations available including memory optimized and compute optimized options

Hardware Configuration
Gen5
up to 80 vCores, up to 408 GB memory
[Change configuration](#)

Save money
Save up to 55% with a license you already own. Already have a SQL Server license? [?](#)

Yes No

License type
SQL server

I confirm that I have a SQL server license with Software Assurance to apply this Azure Hybrid Benefit for SQL Server

vCores [How do vCores compare with DTUs?](#) 2 vCores

Data max size 32 GB 1 TB 32 GB

9.6 GB LOG SPACE ALLOCATED

Cost summary

- Gen5 - General Purpose (GP_Gen5_2)**
Cost per vCore (in USD)
vCores selected **x 2**
Azure Hybrid Benefit discount **- 5%**
- Cost per GB (in USD)**
Max storage selected (in GB) **x 41.6**
- ESTIMATED COST / MONTH** **9 USD**

Deploying an Azure SQL Database via PowerShell/CLI

You can also deploy your database using Azure PowerShell or the Azure CLI. The image below shows the PowerShell example where you are creating a new resource group, and defining an admin called SqlAdmin and then creating a new server, database, and firewall rule.

```
# Connect-AzAccount

# The SubscriptionId in which to create these objects
$SubscriptionId = ''

# Set the resource group name and location for your server
$resourceGroupName = "myResourceGroup-$($Get-Random)"
.setLocation = "westus2"

# Set an admin login and password for your server
$adminSqlLogin = "SqlAdmin"
$password = "ChangeYourAdminPassword1"

# Set server name - the logical server name has to be unique in the system
$serverName = "server-$($Get-Random)"

# The sample database name
$databaseName = "mySampleDatabase"

# The ip address range that you want to allow to access your server
```

```
$startIp = "0.0.0.0"
$endIp = "0.0.0.0"

# Set subscription
Set-AzContext -SubscriptionId $subscriptionId

# Create a resource group
$resourceGroup = New-AzResourceGroup -Name $resourceGroupName -Location $location

# Create a server with a system wide unique server name
$server = New-AzSqlServer -ResourceGroupName $resourceGroupName
-ServerName $serverName
-Location $location
-SqlAdministratorCredentials $(New-Object -TypeName
System.Management.Automation.PSCredential -ArgumentList $adminSqlLogin, $(ConvertTo-
SecureString -String $password -AsPlainText -Force))

# Create a server firewall rule that allows access from the specified IP range

$serverFirewallRule = New-AzSqlServerFirewallRule -ResourceGroupName $resourceGroupName
-ServerName $serverName
-FirewallRuleName "AllowedIPs" -StartIpAddress $startIp -EndIpAddress $endIp

# Create a blank database with an S0 performance level

$database = New-AzSqlDatabase -ResourceGroupName $resourceGroupName
-ServerName $serverName
-DatabaseName $databaseName
-RequestedServiceObjectiveName "S0"
-SampleName "AdventureworksLT"
```

The Azure CLI can also be used to deploy an Azure SQL Database as shown below:

```
#!/bin/bash

# set execution context (if necessary)
az account set --subscription <replace with your subscription name or id>

# Set the resource group name and location for your server
resourceGroupName=myResourceGroup-$RANDOM
location=westus2

# Set an admin login and password for your database
adminLogin=ServerAdmin
password=`openssl rand -base64 16` 

# password=<EnterYourComplexPasswordHere>

# The logical server name has to be unique in all of Azure
servername=server-$RANDOM

# The ip address range that you want to allow to access your DB
startip=0.0.0.0
endip=0.0.0.0
```

```

# Create a resource group
az group create \
--name $resourceGroupName \
--location $location

# Create a logical server in the resource group
az sql server create \
--name $servername \
--resource-group $resourceGroupName \
--location $location \
--admin-user $adminlogin \
--admin-password $password

# Configure a firewall rule for the server

az sql server firewall-rule create \
--resource-group $resourceGroupName \
--server $servername \
-n AllowYourIp \
--start-ip-address $startip \
--end-ip-address $endip

# Create a database in the server
az sql db create \
--resource-group $resourceGroupName \
--server $servername \
--name mySampleDatabase \
--sample-name AdventureworksLT \
--edition GeneralPurpose \
--family Gen4 \
--capacity 1 \

```

Echo random password
echo \$password

Deploying Azure SQL Database Using Azure Resource Manager templates

Another method for deploying resources is as mentioned earlier is using an Azure Resource Manager template. A Resource Manager template gives you the most granular control over your resources, and Microsoft provides a GitHub repository called “Azure-Quickstart-Templates”, which hosts Azure Resource Manager templates that you can reference in your deployments. A PowerShell example of deploying a GitHub based template is shown below:

```

#Define Variables for parameters to pass to template
$ projectName = Read-Host -Prompt "Enter a project name"
$ location = Read-Host -Prompt "Enter an Azure location (i.e. centralus)"
$ adminUser = Read-Host -Prompt "Enter the SQL server administrator username"
$ adminPassword = Read-Host -Prompt "Enter the SQL server administrator password" -AsSecureString
$ resourceGroupName = "${projectName}rg"

#Create Resource Group and Deploy Template to Resource Group
New-AzResourceGroup -Name $resourceGroupName -Location $location

New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName

```

```
-TemplateUri "https://raw.githubusercontent.com/Azure/azure-quickstart-  
templates/master/101-sql-logical-server/azuredeploy.json"  
-administratorLogin $adminUser -administratorLoginPassword $adminPassword  
  
Read-Host -Prompt "Press [ENTER] to continue ..."
```

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

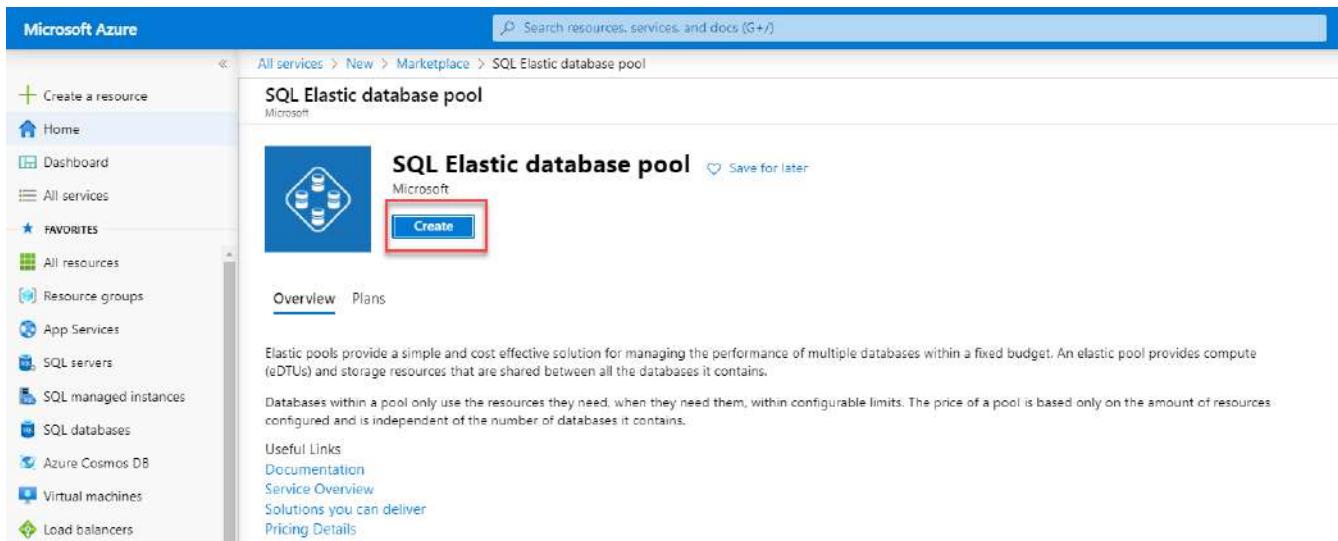
This document bel...

Deploy SQL database elastic pool

Elastic pools are a deployment option in which you purchase Azure compute resources (CPU, memory, and storage) that is then shared among multiple databases defined as belonging to the same pool. An easy comparison to an on-premises SQL Server is that an elastic pool is like a SQL Server instance that has multiple user databases. By using elastic pools, you can easily manage pool resources while at the same time potentially saving costs. Elastic pools also facilitate easy scalability up to the set limits such that if a single database within the pool needs resources due to an unpredictable workload, the resources are there. If the entire pool needs additional resources, a simple slider option within the Azure portal will facilitate scaling the elastic pool up or down.

Creating new elastic pools

Using the Azure portal, click **Create a Resource** and then search for “SQL Elastic database pool” and you will see the screen shown below.



Click **Create** shown in the image above in order to launch the screen shown in the image below.



Create SQL Elastic pool

Microsoft

[Basics](#) [Tags](#) [Review + create](#)

Create a SQL Elastic pool with your preferred configurations. Elastic pools provide a simple and cost effective solution for managing the performance of multiple databases within a fixed budget. Complete the Basic tab, then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Dev-Test-Lab

Resource group * ⓘ

Contoso

[Create new](#)

Elastic pool details

Enter required settings for this pool, including picking a logical server and configuring the compute and storage resources.

Elastic Pool Name *

Enter an elastic pool name

Server * ⓘ

contososql1 (West US)

[Create new](#)

Compute + storage * ⓘ

General Purpose

Gen5, 2 vCores, 32 GB, 0 databases

[Configure elastic pool](#)

Adding a database to an existing pool

Using the Azure portal, locate the pool to which you are adding a database, as showing in the image below.

The screenshot shows the 'Configure' blade for an elastic pool. At the top, there's a 'Feedback' button and a note: 'Looking for basic, standard, premium?'. Below this, there are two columns: 'General Purpose' and 'Business Critical'. The 'General Purpose' column includes details: 'Scalable compute and storage options', '500 - 20,000 IOPS', '2-10 ms latency', and 'Starting at 199.13 USD / month'. The 'Business Critical' column includes details: 'High transaction rate and high resiliency', '5,000 - 204,800 IOPS', '1-2 ms latency', and 'Starting at 1056.69 USD / month'. At the bottom, there are tabs for 'Pool settings', 'Databases' (which is highlighted with a red box), and 'Per database settings'. Below these tabs, there's a 'Revert selected' button and a search bar 'Search to filter databases...'. A note says: 'Currently, there are no databases selected to be added to the pool. To add databases, click 'Add databases' above.'

The image below shows the process for selecting which database(s) you wish to add to the pool.

Add databases

Select all Selected/Total databases 1/4

Search to filter databases...

Database name	Pricing tier	Database size
<input checked="" type="checkbox"/> AdventureWorks2014	General Purpose: Serverless, Gen5, ...	272 MB
<input type="checkbox"/> AdventureWorksDW2014	General Purpose: Serverless, Gen5, ...	111 MB

Click **Apply** on the screen shown in the image below.

Configure

Feedback

Looking for basic, standard, premium?

General Purpose	Business Critical
Scalable compute and storage options 500 - 20,000 IOPS 2-10 ms latency Starting at 187.77 USD / month	High transaction rate and high resiliency 5,000 - 204,800 IOPS 1-2 ms latency Starting at 1011.32 USD / month

Pool settings Databases Per database settings

+ Add databases Revert selected

Search to filter databases...

Database name	Pricing tier	Data space used
<input type="checkbox"/> AdventureWorks2014	General Purpose: Serverless, Gen5, 1 vCore	272 MB

Apply

Click **Apply** one more time and the database will be added to the elastic pool.

Managing pool resources

The Azure portal delivers a wealth of information regarding the state and health of the elastic pool. You can view resource utilization and see which database is consuming the most resources. This information can be helpful for diagnosing performance issues or identify a database that might not be a good fit for the pool, such as when one database is consuming the vast majority of pool resources. The image below shows an elastic pool with even resource utilization.

contoso-eastus-elastic-pool (contoso-server/contoso-eastus-elastic-pool)

Overview

Resource group (resource-group) : resource-group

Status : Ready

Location : East US

Subscription (My Subscription) : My Subscription

Subscription ID : 63248ac6-5235-4e27-b768-1172c010efaa

Tags (tags) : Click here to add tags

Show data for last: 1 hour 24 hours 7 days

Resource utilization (contoso-eastus-elastic-pool)

Aggregation type: Max

Blob storage (50 MB)

Used space: 50 MB

Allocated space: 50 MB

Maximum storage size: 4.88 GB

1% USED SPACE

Notifications (0)

All Alerts (0) Recommendations (0) Info (0)

There are no notifications to display.

All Security (0) Performance (0) Recovery (0)

If you need to adjust the pool to decrease or increase resources allocated to the pool, you can make that change via the **Configure** option in the **Pool settings** section of the **Elastic Pool** management blade.

From that blade, you can quickly and easily adjust:

- Pool size including DTUs, vCores, and storage size.
- Service Tier
- Resources per database
- Which databases are included in the pool, by adding or removing them.

As shown in the image below, you can adjust numerous settings in the Elastic Pool. Many of these changes can be made online, including the min and max DTUs or vCores per database. **You can change the size of total size of the pool or add and remove databases from the pool as needed. Active connections will be dropped as the resizing completes.**

contoso-eastus-elastic-pool (contoso-server/contoso-eastus-elastic-pool) | Configure

Pool settings

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

Service tier: Basic (For less demanding workloads)

eDTUs: 50 DTUs

Data max size: 4.88 GB

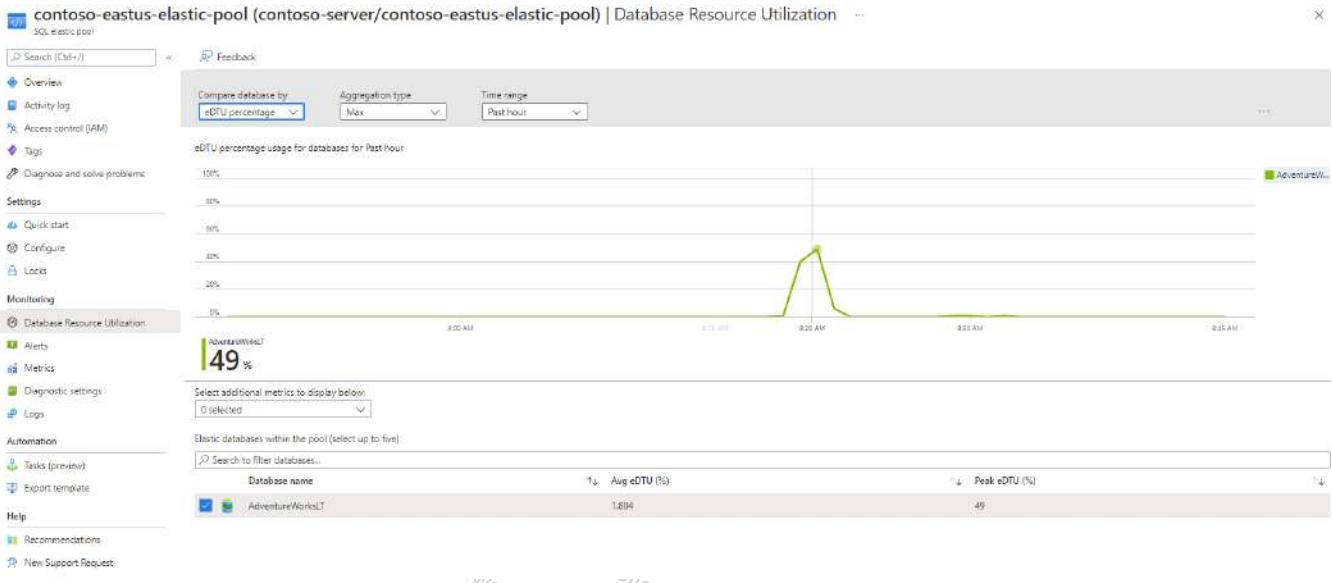
Cost summary

Cost per DTU (in USD): 1.50

DTUs selected: x 50

ESTIMATED COST / MONTH: 75.02 USD

Probably the most useful feature is the ability to monitor Database Resource Utilization, as shown in the image below. This feature allows you to easily see how databases are performing within the pool.



An elastic pool is a good fit for multi-tenant databases where each tenant has its own copy of the database. Balance the workload across databases so as not to allow one database to monopolize all the pool's resources.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Understand SQL database hyperscale

Azure SQL Database has been limited to 4 TB of storage per database for many years. This restriction is due to a physical limitation of the Azure infrastructure. Azure SQL Database Hyperscale changes the paradigm and allows for databases to be 100 TB or more. Hyperscale introduces new horizontal scaling techniques to add compute nodes as the data sizes grow. The cost of Hyperscale is the same as the cost of Azure SQL Database; however, there's a per terabyte cost for storage. You should note that once an Azure SQL Database is converted to Hyperscale, you can't convert it back to a "regular" Azure SQL Database. Hyperscale is the ability for an architecture to scale appropriately as demanded.

Azure SQL Database Hyperscale is a great option for most business workloads as it provides great flexibility and high performance with independently scalable compute and storage resources.

Hyperscale 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, storage capacity can be smoothly scaled out as far as needed.

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 uses Azure 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.

Benefits

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're billed only for the capacity you use. For read-intensive workloads, the Hyperscale service tier provides rapid scale-out by provisioning extra 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 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. Hyperscale also provides fast database restores which runs in minutes rather than hours or days.

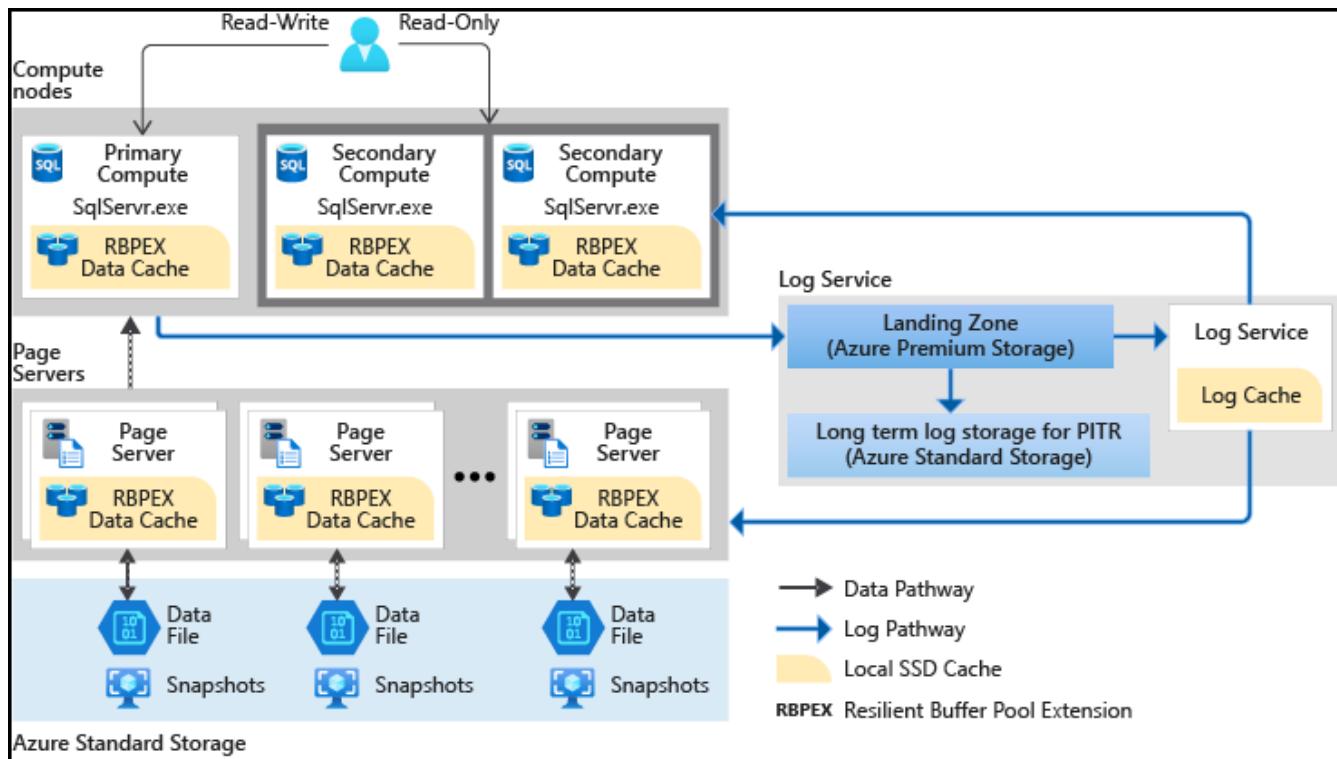
Hyperscale provides rapid scalability based on your workload demand.

- **Scaling Up/Down** – You can scale up the primary compute size in terms of resources like CPU and memory, and then scale down, in constant time. Because the storage is shared, scaling up and scaling down isn't linked to the volume of data in the database.
- **Scaling In/Out** – You also get the ability to provision one or more compute replicas that you can use to serve your read requests. This means that you can use extra compute replicas as read-only replicas to offload your read workload from the primary compute. In addition to read-only, these replicas also serve as hot-standbys to failover from the primary.

Provisioning of each of these extra compute replicas can be done in constant time and is an online operation. You can connect to read-only compute replicas by setting the *ApplicationIntent* argument on your connection string to

ReadOnly. Any connections with the **ReadOnly** application intent are automatically routed to one of the read-only compute replicas.

Hyperscale separates the query processing engine from the components that provide long-term storage and durability for the data. This architecture provides the ability to smoothly scale storage capacity as far as needed (initial target is 100 TB), and the ability to scale compute resources rapidly.



Security considerations

Security for Hyperscale service tier shares the same great capabilities as the other Azure SQL Database tiers. They're protected by the layered defense-in-depth approach as shown in the picture below, and moves from the outside in:

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!



- **Network Security** is the first layer of defense and uses IP firewall rules to allow access based on the originating IP address and Virtual Network firewall rules to allow the ability to accept communications that are sent from selected subnets inside a virtual network.
- **Access Management** is provided through the below authentication methods to ensure a user is whom they claim to be:
 - SQL Authentication
 - Azure Active Directory Authentication
 - Windows Authentication for Azure AD Principals (Preview)

Azure SQL Database Hyperscale also supports 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).



- **Threat Protection** abilities in auditing and threat detection capabilities. SQL Database and SQL Managed Instance auditing tracks database activities and helps maintain compliance with security standards by

recording database events to an audit log in a customer-owned Azure storage account. Advanced Threat Protection can be enabled per server for an extra fee and analyzes your logs to detect unusual behavior and potentially harmful attempts to access or exploit databases. Alerts are created for suspicious activities such as SQL injection, potential data infiltration, and brute force attacks or for anomalies in access patterns to catch privilege escalations and breached credentials use.

- **Information Protection** is provided in the following various ways:
 - Transport Layer Security (Encryption-in-transit)
 - Transparent Data Encryption (Encryption-at-rest)
 - Key management with Azure Key Vault
 - Always Encrypted (Encryption-in-use)
 - Dynamic data masking

Performance considerations

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

Hyperscale provides the following performance capabilities:

- Nearly instantaneous database backups (based on file snapshots stored in Azure Blob storage) regardless of size with no IO effect on compute resources.
- Fast database restores (based on file snapshots) in minutes rather than hours or days (not a size of data operation).
- Higher overall performance due to higher transaction log throughput and faster transaction commit times regardless of data volumes.
- Rapid scale out - you can provision one or more read-only replicas 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 when needed, and then scale the compute resources back down when not needed.

NOTE: SQL Database Hyperscale does not support the following features:

- SQL Managed Instance
- Elastic Pools
- Geo-replication
- Query Performance Insights

Deploying Azure SQL Database Hyperscale

To deploy Azure SQL Database with the Hyperscale tier:

1. Browse to the **Select SQL Deployment** option page.

2. Under **SQL databases**, leave **Resource type** set to **Single database**, and select **Create**.

How do you plan to use the service?

- SQL databases**: Best for modern cloud applications. Hyperscale and serverless options are available. Resource type: Single database (highlighted with a red arrow). **Create** button (highlighted with a red box).
- SQL managed instances**: Best for most migrations to the cloud. Lift-and-shift ready. Resource type: Single instance.
- SQL virtual machines**: Best for migrations and applications requiring OS-level access. Lift-and-shift ready. Image dropdown.
- Database server**: Database servers are used to manage groups of single databases and elastic pools. **Featured capabilities:**
 - Hyperscale storage (up to 100TB)
 - Serverless compute
 - Easy management
- Elastic pool**: Elastic pools provide a cost-effective solution for managing the performance of multiple databases with variable usage patterns. **Featured capabilities:**
 - Resource sharing for cost optimization
 - Simplified performance management

3. From the **Basics** tab of the **Create SQL Database** page, select the desired subscription, resource group, and database name.
4. Select the **Create new** link for the **Server**, and fill out the new server information, such as server name, server admin login and password, and location.
5. Under **Compute + storage**, select the **Configure database** link.

[Home > SQL databases >](#)

Create SQL Database

Microsoft

[Basics](#)[Networking](#)[Security](#)[Additional settings](#)[Tags](#)[Review + create](#)

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#)

i Did you know that new users in Azure can create a free Azure SQL Database and use it for 12 months using Azure free account? [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Contoso Subscription

Resource group * ⓘ

contoso-rg

[Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name *

[Enter database name](#)

Server * ⓘ

contoso-sqlserver (West US 2)

[Create new](#)

Want to use SQL elastic pool? * ⓘ

 Yes No

Compute + storage * ⓘ

General Purpose

Gen5, 2 vCores, 32 GB storage, zone redundant disabled

[Configure database](#)

6. For Service tier, select Hyperscale.

No unauthorized copies allowed!

This document belongs to Ramchand Jillella.
No unauthorized copies allowed!

[Home](#) > [SQL databases](#) > [Create SQL Database](#) >

Configure

[Feedback](#)

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

Service tier

Hyperscale (On-demand scalable storage)

Databases originally created in Hyperscale

vCore-based purchasing model

General Purpose (Scalable compute and storage options)

Hyperscale (On-demand scalable storage)

Business Critical (High transaction rate and high resiliency)

DTU-based purchasing model

Basic (For less demanding workloads)

Standard (For workloads with typical performance requirements)

Premium (For IO-intensive workloads)

[Change configuration](#)

Compute Hardware

Select the hardware configuration based on confidential computing hardware depends on the service tier selected.

Hardware Configuration

Save money

Already have a SQL Server License? Save with a license you already own with Azure Hybrid Benefit. Actual savings may vary based on region and performance tier. [Learn more](#)

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

2

7. Under **Hardware Configuration**, select the **Change configuration** link. Review the available hardware configurations and select the most appropriate configuration for your database. For this example, we'll select the **Gen5** configuration.
8. Select OK to confirm the hardware generation.
9. Optionally, adjust the **vCores** slider if you would like to increase the number of vCores for your database. For this example, we'll select **2 vCores**.
10. Adjust the High-Availability Secondary Replicas slider to create one High Availability (HA) replica. Select **Apply**.
11. Select **Next: Networking** at the bottom of the page.

This document belongs to srujanasreddi@gmail.com
No unauthorized copies are allowed!

Want to use SQL elastic pool? * Yes No

Compute + storage * Hyperscale
Gen5, 2 vCores
[Configure database](#)

Backup storage redundancy

Choose how your PITR and LTR backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

Backup storage redundancy Locally-redundant backup storage - Preview
 Zone-redundant backup storage - Preview
 Geo-redundant backup storage

⚠️ Selected value for backup storage redundancy is Geo-redundant backup storage. Note that database backups will be geo-replicated to the paired region. [Learn more](#)

ⓘ Your use of either of the Preview backup storage redundancy options (ZRS and LRS) is governed by the agreement under which you obtained

[Review + create](#) [Next : Networking >](#)

12. For **Firewall rules** on the **Networking** tab, set **Add current client IP address** to **Yes**. Leave **Allow Azure services and resources to access this server** set to **No**.
13. Select **Next: Security** at the bottom of the page.
14. On the **Review + create** tab, select **Create**.



This document belongs to Ramchand Jillella.
Unauthorized copies allowed!

Examine SQL managed instance

While many organizations initially migrate to Azure using IaaS offerings, the platform as a service (PaaS) service offering allows for extra benefits. **Also, you no longer have to install or patch SQL Server as that is performed by the service.** Consistency checking, and backups are also part of the managed service, including security and performance tools that are part of the PaaS offerings.

Azure SQL Managed Instance is a fully functional SQL Server instance that is almost 100% compatible with your on-premises ecosystem. It includes features like SQL Agent, access to tempdb, cross-database query and common language runtime (CLR). The service uses the same infrastructure as Azure SQL Database and all the benefits of the PaaS service such as automatic backups, automatic patching, and built-in high availability.

Azure SQL Managed Instance features

Azure SQL Managed Instance allows for easy migration paths for existing applications by allowing restores from on-premises backups. Unlike Azure SQL Database, which is designed around single database structures, SQL Managed Instance provides an entire SQL Server instance, allowing up to 100 databases, and providing access to the system databases. SQL Managed Instance provides other features that aren't available in Azure SQL Database, including cross-database queries, common language runtime (CLR) and along with the msdb system database, it allows the use of SQL Agent.

Options

There are two service tiers available when creating an Azure SQL Managed Instance, and they're the same as Azure SQL Database vCore model (managed instance is purchased using the vCore model), Business Critical and General Purpose. There are minimal functionality differences between the two tiers—the main two are that **Business Critical includes In-Memory OLTP and offers a readable secondary**, neither of which is available with the General Purpose tier. Both tiers offer the same levels of availability and allow for independent configuration of storage and compute.

Link feature (preview)

Link feature provides hybrid capability of replicating databases from SQL Server instances to Azure SQL Managed Instance. The link feature replicates data using distributed availability groups available on Always On availability group technology. Transaction log records are replicated as part of distributed availability groups.

The transaction log records on the primary instance can't be truncated until they've been replicated to the secondary instance. Regular transaction log backups reduce the risk of running out of space on your primary instance.

Link feature can also be used as a hybrid disaster recover solution, where you can fail over your SQL Server databases hosted anywhere to a database running on SQL Managed Instance. Likewise, you can use link feature to provide a read-only secondary database in SQL Database SQL Managed Instance to offload intensive read-only operations.

For more information about how to configure link feature for Azure SQL Managed Instance, see [Prepare environment for link feature - Azure SQL Managed Instance](#).

Instance pool (preview)

Instance pool provides a cost-efficient way to migrate smaller SQL Server instances to the cloud. When migrating to Azure, instead of consolidating smaller databases into larger managed instance, which requires extra governance and security planning, instance pools allow you to pre-provision your resources based on your total migration resources and requirements.

The instance pool feature provides a fast deployment time of up to five minutes, which is a good option for scenarios where deployment duration is important. Also, all instances in a pool share the same virtual machine, and the total IP allocation is independent of the number of instances deployed.

To learn how to deploy an instance pool for SQL Managed Instance, see [Deploy Azure SQL Managed Instance to an instance pool](#).

High availability

Because Azure SQL Managed Instance is backed by the PaaS Service, it has high availability baked into the product. A standalone SQL Managed Instance offers a 99.99% Service Level Agreement (SLA) which guarantees at most 52.60 minutes of downtime per year. The architecture is the same as Azure SQL Database with General Purpose, which uses storage replication for availability, and Business critical using multiple replicas.

Backups

Automatic backups are also automatically configured for Azure SQL Managed Instance. One key difference between Azure SQL Managed Instance and Azure SQL Database is that with MI you can manually make a copy-only backup of a database. You must back up to a URL, as access to the local storage isn't permissible. You can also configure long-term retention (LTR) for retaining automatic backups for up to 10 years in geo-redundant Azure blob storage.

Database backups occur on the same schedule as with Azure SQL Database. These schedules aren't adjustable.

- **Full** – Once a week
- **Differential** – Every 12 hours
- **Transaction Log** – Every 5-10 minutes depending on transaction log usage

Restoring a database to an Azure SQL Managed Instance is also similar to the process with Azure SQL Database. You can use:

- Azure portal
- PowerShell
- Azure CLI

However, there are some limitations when restoring. **In order to restore from one instance to another, both instances must reside within the same Azure subscription and the same Azure region. You also can't restore the entire managed instance, only individual databases within the SQL Managed Instance itself.**

As with Azure SQL Database, you can't restore over an existing database. You need to drop or rename the existing database prior to restoring it from backup. Since SQL Managed Instance is a fully functional SQL Server instance, **you can execute a RESTORE command whereas with Azure SQL Database that isn't possible.** However, since it's a PaaS service, there are some limitations, such as:

- You must restore from a URL endpoint. You don't have access to local drives.

- You can use the following options (in addition to specifying the database):
 - FILELISTONLY
 - HEADERONLY
 - LABELONLY
 - VERIFYONLY
- Backup files containing multiple log files can't be restored
- Backup files containing multiple backup sets can't be restored
- Backups containing In-Memory/FILESTREAM can't be restored

By default, the databases in a managed instance are encrypted using Transparent Data Encryption (TDE) with a Microsoft managed key. In order to take a user-initiated copy only backup, you must turn off TDE for the specific database. If a database is encrypted, you can restore it, however, you'll need to ensure that you have access to either the certificate or asymmetric key that was used to encrypt the database. If you don't have either of those two items, you won't be able to restore the database to a SQL Managed Instance.

Disaster recovery

Azure SQL Managed Instance offers auto-failover groups as a means to implement disaster recovery. This feature protects the entire managed instance and all of the databases contained within it, not just specific databases. **This process asynchronously replicates data from the Azure SQL Managed Instance to a secondary;** however, it's currently limited to the paired Azure region of the primary copy, and only one replica is allowed.

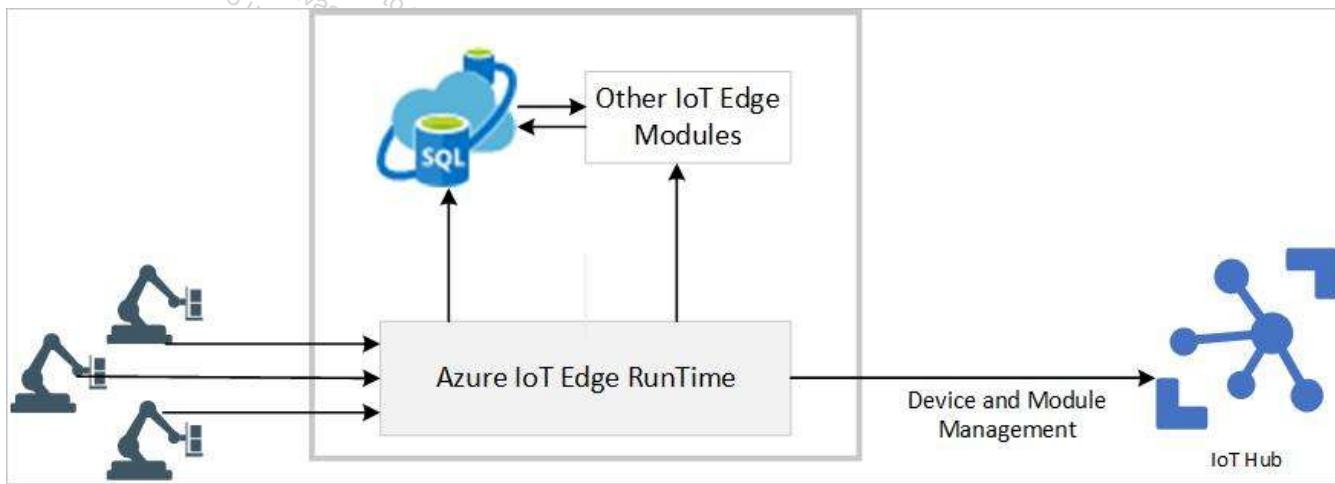
Much like Azure SQL Database, auto-failover groups offer read-write and read-only listener endpoints, which facilitate easy connection string management. **If there is a failover, application connection strings will automatically be routed to the appropriate instance.** While fairly consistent with Azure SQL Database, these endpoints follow a slightly different format, `<fog-name>.zone_id.database.windows.net` whereas **Azure SQL Database is in the `<fog-name>.secondary.database.windows.net` format.**

Each managed instance, primary and secondary, must be within the same DNS zone. This placement will ensure that the same multi-domain certificate can be used for client connection authentication between either of the two instances in the same failover group. You can specify a "DNS Zone Partner" through various methods such as the Azure portal, PowerShell, or Azure CLI.

To learn the new features for Azure SQL Managed Instance, see [What's new in Azure SQL Managed Instance?](#)

Describe SQL Edge

Many organizations have substantial investments in IoT infrastructure. A typical IoT solution architecture includes IoT devices responsible for reading environmental sensors to generate customer data. Commonly, this data gets processed on-site using Edge devices. In addition, an IoT Edge device can run Docker compatible containers containing custom business logic or light-weight versions of cloud services such as Azure Stream Analytics, Azure Machine Learning, Azure Functions, Azure SQL, and more. The benefit to IoT Edge is that processing happens on the local network resulting in a quicker feedback loop should any action need to be taken, at the same time minimizing cloud processing and bandwidth costs.



Azure SQL Edge is an optimized relational database engine purposefully designed for IoT workloads. It provides capabilities to stream, process, and analyze relational and non-relational data such as JSON, graph, and time-series data. Azure SQL Edge is built on the latest version of the SQL Server Database Engine – the same engine that serves as the foundation of SQL Server and Azure SQL. Azure SQL Edge brings T-SQL programming, industry-leading performance, security, and query processing capabilities to the Edge.

Benefits

Familiar T-SQL syntax and tooling

SQL Developers and administrators can continue to leverage familiar T-SQL syntax and tooling since Azure SQL Edge is based on the SQL Server Database Engine. Tooling available includes the Azure portal, SQL Server Management Studio, Azure Data Studio, Visual Studio Code, and SQL Server Data Tools in Visual Studio.

Portability

Azure SQL Edge is a containerized version of the SQL Server Database Engine optimized for IoT. Azure SQL Edge is deployable to various Windows and Linux-based servers capable of running the IoT Edge runtime, ranging from powerful full-fledged servers to smaller ARM-based devices.

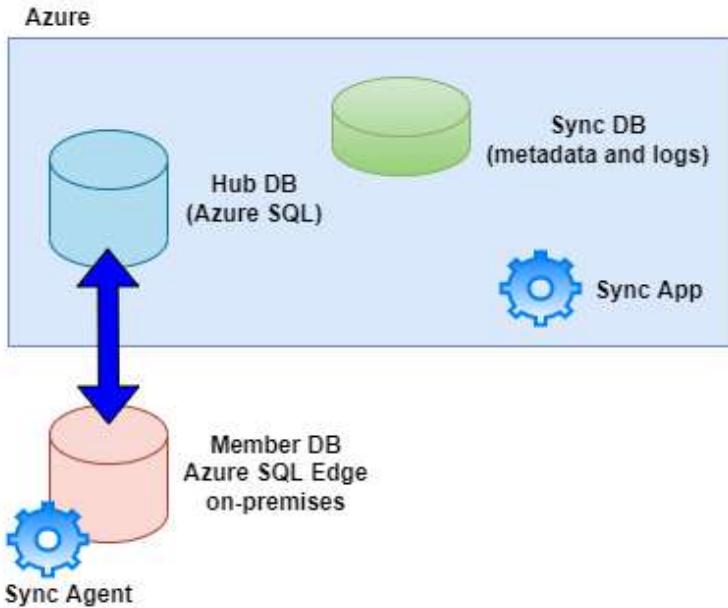
Support for multiple connection states and data sync

In IoT, internet connectivity isn't always possible or reliable. Therefore, IoT Edge modules need to support all states of connectivity. Azure SQL Edge supports connected, disconnected, and hybrid semi-connected scenarios.

Incremental data synchronization is possible with the Azure SQL Data Sync service and configuring sync groups

to synchronize the tables you choose bi-directionally across multiple databases in Azure SQL and SQL Server instances.

The diagram below depicts the synchronization process. The synchronization process uses a sync agent on the Azure SQL Edge to sync data with the Hub database. From the Hub perspective, the synchronization process is driven by a Sync app guided by details available in the Sync database, where the synchronization metadata and logs get stored.



Built-in data streaming and machine learning

Azure SQL Edge has built-in support for data streaming to and from multiple inputs and outputs. This functionality borrows the same technology that powers Azure Stream Analytics and allows introspection of incoming time-series data using anomaly detection, time-windowing, aggregation, and filtering. Azure SQL Edge also has T-SQL functions that support querying time-series data. Furthermore, Azure SQL Edge supports machine learning inference and the `PREDICT` statement.

Security considerations

Security on Azure SQL Edge brings data encryption, classification, and access controls from the SQL Server Database Engine. In addition, Azure SQL Edge provides row-level security, dynamic data masking, and transparent data encryption (TDE) as an extra security benefit. It's also beneficial to encrypt any backup files created using a certificate or asymmetric key.

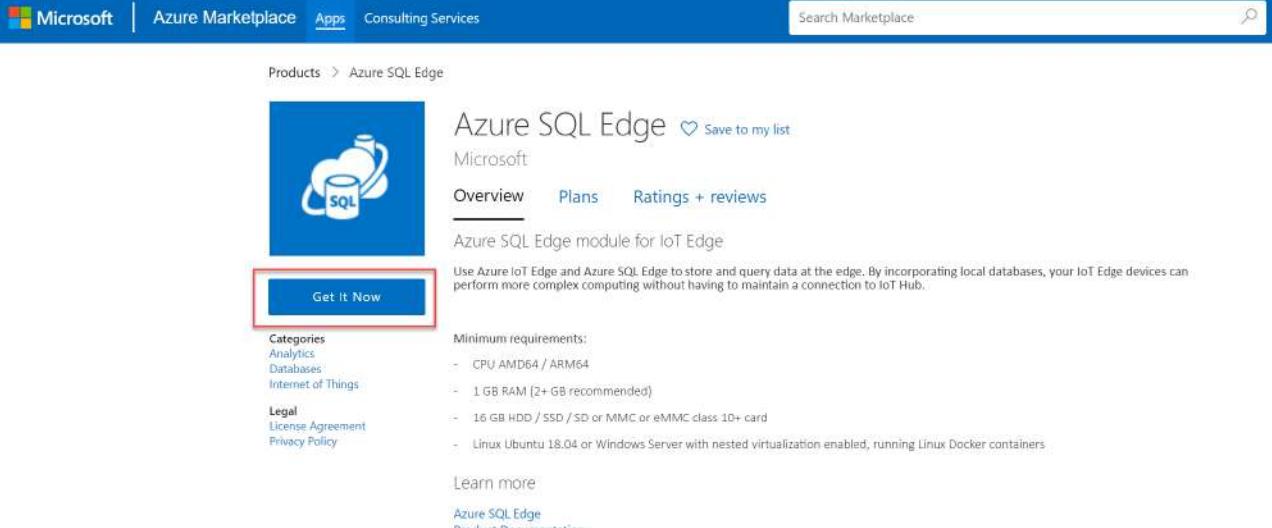
As for network transport, Azure SQL Edge utilizes transport layer security (TLS) and certificates to encrypt all communication. Lastly, Microsoft Defender for IoT provides a centralized and unified security solution to discover and identify IoT devices, vulnerabilities, and threats. As with any data-related solution, it's also prudent to ensure that database users are granted the least privilege on database objects.

Deploying Azure SQL Edge from the Azure Marketplace

Azure SQL Edge is available in the Azure Marketplace with two plans, Azure SQL Edge Developer (for development only, limited to 4 cores and 32 GB of memory), and Azure SQL Edge (for production, limited to 8 cores and 64 GB of memory).

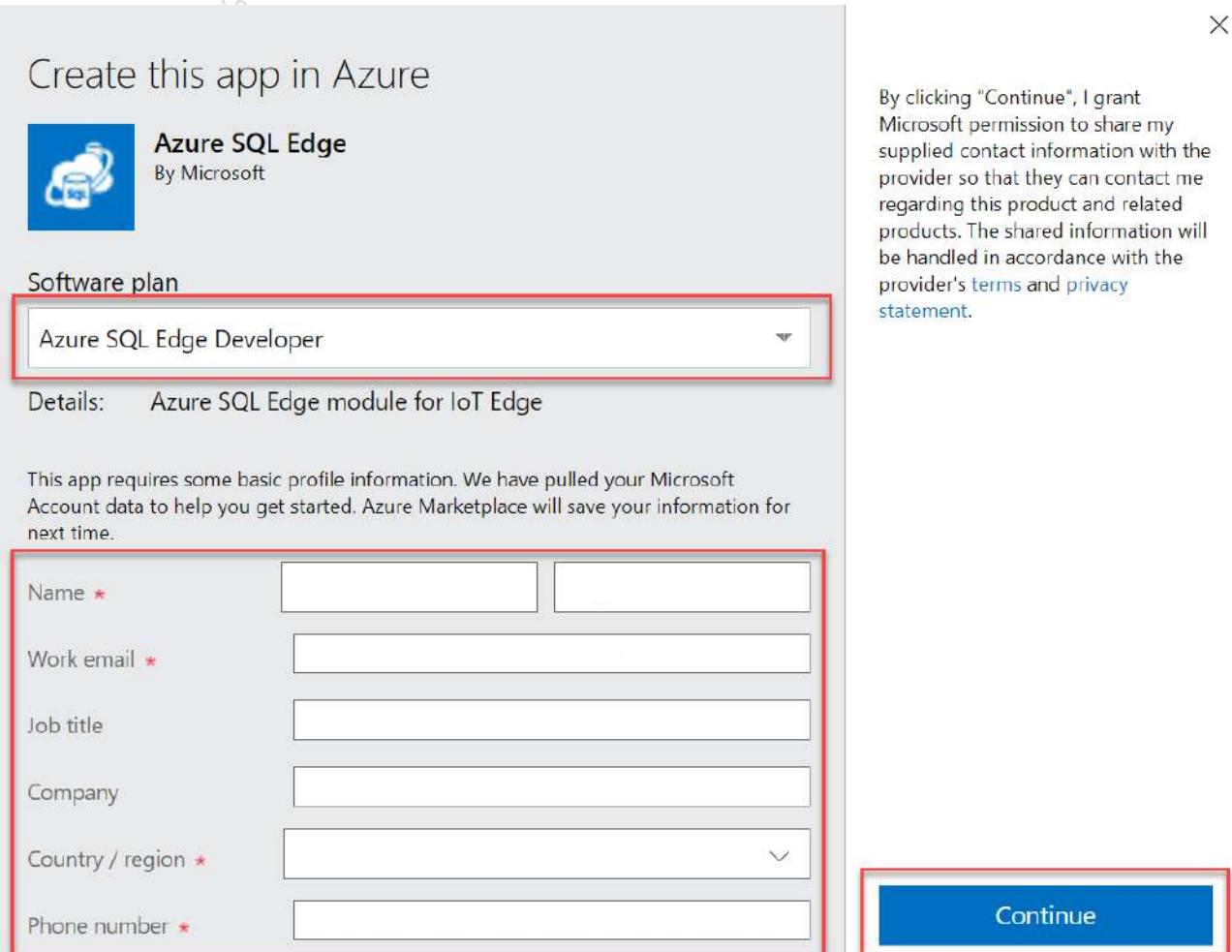
As a pre-requisite to deploy Azure SQL Edge, you need to have an IoT Hub provisioned with at least one IoT Edge device. In this example, an IoT Hub named **org-iot-hub** and a Linux-based IoT Edge device called **iot-edge-device-1** have been pre-provisioned.

1. Locate the **Azure SQL Edge module in the Azure Marketplace**, and select the **Get It Now** button.



The screenshot shows the Azure Marketplace interface. At the top, there are navigation links for Microsoft, Azure Marketplace, Apps, Consulting Services, and a search bar labeled 'Search Marketplace'. Below this, a breadcrumb trail shows 'Products > Azure SQL Edge'. The main content area features a large thumbnail for 'Azure SQL Edge' with a blue background and a white icon of a teapot and a cup. To the right of the thumbnail, the product name 'Azure SQL Edge' is displayed along with a 'Save to my list' button. Below the thumbnail, there are tabs for 'Overview' (which is selected), 'Plans', and 'Ratings + reviews'. A brief description of the Azure SQL Edge module for IoT Edge follows, mentioning its use for storing and querying data at the edge. A section for 'Minimum requirements' lists the following: CPU AMD64 / ARM64, 1 GB RAM (2+ GB recommended), 16 GB HDD / SSD / SD or MMC or eMMC class 10+ card, and Linux Ubuntu 18.04 or Windows Server with nested virtualization enabled, running Linux Docker containers. Below this, there are links for 'Learn more', 'Azure SQL Edge', and 'Product Documentation'. On the left side, there are categories like Analytics, Databases, Internet of Things, Legal, License Agreement, and Privacy Policy.

2. In the modal form, select the desired software plan SKU. In this example, **Azure SQL Edge Developer** is chosen. Next, fill in any other profile information required by the form and select **Continue**.



The screenshot shows a modal window titled 'Create this app in Azure'. At the top, it displays the product information: 'Azure SQL Edge' by Microsoft. Below this, a 'Software plan' dropdown is set to 'Azure SQL Edge Developer' and is highlighted with a red box. The 'Details' section below it shows the description 'Azure SQL Edge module for IoT Edge'. A note states: 'This app requires some basic profile information. We have pulled your Microsoft Account data to help you get started. Azure Marketplace will save your information for next time.' The profile information section, which includes fields for Name, Work email, Job title, Company, Country / region, and Phone number, is also highlighted with a red box. On the right side of the modal, there is a note about granting Microsoft permission to share contact information, followed by a 'Continue' button which is also highlighted with a red box.

3. In the **Target Devices for IoT Edge Module** screen, enter the IoT Edge Device Name value manually or use the **Find Device** functionality to locate the Edge device from the selected IoT Hub. In this example, the Edge device name is **iot-device-edge-1**. Next, select the **Create** button.

Home >

Target Devices for IoT Edge Module

Microsoft

Subscription * ⓘ

Azure Subscription

IoT Hub * ⓘ

org-iot-hub

Deploy to a device Deploy at Scale

IoT Edge Device Name * ⓘ

iot-edge-device-1

Find Device

By deploying this module, I agree to the provider's [terms of use](#) and [privacy policy](#) and understand that the rights to use this product do not come from Microsoft, unless Microsoft is the provider. Use of Azure Marketplace is governed by separate terms.

Create

4. On the **Set modules on device** blade, choose the **AzureSQLEdge** item under IoT Edge Modules.

Set modules on device: iot-edge-device-1

...
org-iot-hub[Modules](#) [Routes](#) [Review + create](#)

Container Registry Credentials

You can specify credentials to container registries hosting module images. Listed credentials are used to retrieve images from the registry.

NAME	ADDRESS
<input type="text" value="Name"/> ...	<input type="text" value="Address"/>

IoT Edge Modules

IoT Edge modules are Docker containers deployed to IoT Edge devices. They can communicate with other modules up to 100 per second if no other updates are happening in the IoT Hub. [Learn more](#)

NAME	DESIRED STATUS
AzureSQLEdge	running

Send usage data to Microsoft to help improve our products and services. Read our [privacy statement](#) to learn more.

5. On the **Update IoT Edge Module** blade, select the **Environment Variables** tab. Next, replace the SQL Edge admin account password by setting the value for the **MSSQL_SA_PASSWORD** variable. Optionally add configuration options under the **Container Create Options** tab. Once complete, select the **Update** button.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...



Update IoT Edge Module

Specify the settings for an IoT Edge custom module.
[Learn more](#)

IoT Edge Module Name *

Module Settings

Environment Variables

Container Create Options

Module Twin Settings

Environment variables provide supplemental information to a module facilitating the configuration process.

NAME	TYPE	VALUE
ACCEPT_EULA	Text	Y
MSSQL_SA_PASSWORD	Text	<Default_MSSQL_SA_Password>
MSSQL_LCID	Text	1033
MSSQL_COLLATION	Text	SQL_Latin1_General_CI_AS
Variable name	Text	Variable value

Update

Cancel

6. Returning to the **Set modules on device** blade, optionally configure message routing for the module beneath the **Routes** tab. Once complete, select **Review + create** and **Create** once more on the validation screen.

[Home](#) > [Target Devices for IoT Edge Module](#) >

Set modules on device: iot-edge-device-1

org-iot-hub

[Modules](#) [Routes](#) [Review + create](#)

Container Registry Credentials

You can specify credentials to container registries hosting module images. Listed credentials are used to retrieve modules with a matching URL. The Edge Agent will report error code 500 if it can't find a container registry setting for a module.

NAME	ADDRESS	USER NAME	PASSWORD
<input type="text" value="Name"/>	<input type="text" value="Address"/>	<input type="text" value="User name"/>	<input type="text" value="Password"/>

IoT Edge Modules

IoT Edge modules are Docker containers deployed to IoT Edge devices. They can communicate with other modules or send data to the IoT Edge runtime. Modules on devices count toward IoT Hub quota limits based on tier and units. For example, for S1 tier, modules can be set 10 times per second if no other updates are happening in the IoT Hub. [Learn more](#)

Add Runtime Settings

NAME	DESIRED STATUS	
AzureSQLEdge	running	



Send usage data to Microsoft to help improve our products and services. Read our [privacy statement](#) to learn more. See [details](#) of what data is collected.

[Review + create](#)

< Previous

Next: Routes >

7. The IoT Edge device screen will display. Wait a few moments, and the device's reported module list now displays **AzureSQLEdge** in a running state. If the module's startup isn't complete, it will temporarily indicate an error state – wait a few minutes and refresh.
 8. Use your desired **connection method** and begin using Azure SQL Edge!
- document below*

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

You need to migrate a set of databases that use distributed transactions from on-premises SQL Server. Which of the following options should you choose?

- Azure SQL Database
- Azure SQL Database Hyperscale
- Azure SQL Managed Instance

Check Answers

Multiple choice

You're building a new cloud database that you expect to grow to 50 TB. Which is the best option for your database?

- Azure SQL Managed Instance
- Azure SQL Database Serverless
- Azure SQL Database Hyperscale

Check Answers

Multiple choice

You're building a database for testing purposes that will be used less than 8 hours a day. It's expected to be 20 GB in size. What is your most cost-effective option?

- Azure SQL Database Serverless
- Azure SQL Database Elastic Pools
- Azure SQL Managed Instance

Check Answers

Multiple choice

How often do differential backups occur with Azure SQL managed instance?

- Every 1 hour
- Every 12 hours
- Every 24 hours

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

In this module, you learned about the platform as a service options for SQL Server on Azure. Azure SQL Database is a flexible platform, well suited for software as service applications and has many options for large databases, multiple databases, or development environments that don't need to be online 24x7. Azure SQL Managed Instance is a PaaS version of SQL Server that allows you to easily move your on-premises environments into a managed service. SQL Managed Instance also supports many server-level capabilities that aren't available with Azure SQL Database.

Now that you've reviewed this module, you should be able to:

- Understand PaaS provisioning and deployment options
- Understand elastic pools and hyperscale features
- Examine SQL Managed Instances
- Understand SQL Edge

Introduction

Understanding the options and methods for migrating on-premises SQL databases to Azure are critical to ensure a successful migration. As part of the planning, it is important to follow the necessary steps that you, as a database administrator, must follow carefully. Selecting the proper compatibility level, understanding Azure preview features, and describing the available migration methods are crucial to ensure the success of your migration.

This module will also provide concrete information on the various options when you are performing a migration.

Learning objectives

At the end of this module, you will be able to:

- Understand how compatibility level affects query optimizer behaviors.
- Understand how Azure preview feature process works.
- Describe the various options when you are performing a migration.

Understand compatibility level

Historically, software vendors who build software for SQL Server have certified their software to run on a specific version of the database engine. For example, SharePoint 2016 was only certified to run on SQL Server 2014. This process, called compatibility certification, allows for an application to run on the latest release of SQL Server, while maintaining its vendor supported compatibility level.

SQL Server compatibility level has always been a database level setting. Setting compatibility level to a specific version allows for specific T-SQL keywords to be used as it also determines certain query optimizer behaviors. For example, if you had a database at a specific compatibility level and migrated it to SQL Server 2019, the execution plan shapes and query syntax should remain the same as they did originally before the migration, if it is a supported release.

The database engine version for Azure SQL Database and Azure SQL Managed Instance are not comparable with SQL Server internal build numbers, but they do refer to the same compatibility level.

You can check the compatibility level of your databases by executing the query as shown below:

```
SELECT name, compatibility_level FROM sys.databases;
```

Support Policy for SQL Server

Microsoft has a generous support policy for SQL Server. **Releases are supported for five years in primary support, and then five additional years in extended support.** During the first five years, Microsoft updates all releases with enhanced capabilities, closes feature gaps, and addresses performance, functional, and security bugs. After a release moves into extended support, Microsoft will only address security bugs.

There are many benefits to running on the latest release of SQL Server, including enhancements in the following categories:

- Performance
- Security
- Availability
- Query functionality

These benefits are further improved by the one to two-year release cadence of SQL Server, and the nature of the Azure SQL Database services, **which means it never needs to be patched or upgraded where new features are added and fixes are applied automatically.**

Microsoft has recommended that application vendors certify applications at a specific compatibility level, instead of for a particular software version. This approach will help customers to take advantage of newer releases of SQL Server but maintain vendor support for applications.

Microsoft includes query plan shape protection, which means your query execution plans and their performance should be nearly the same (on similar hardware). This feature removes one of the main risks of upgrading SQL Server: optimizer changes that causes degradation in query performance. **Microsoft still recommends upgrading to a newer compatibility level when possible but will support databases on older compatibility levels as long as the release of SQL Server that you are running on is a supported release of SQL Server.**

Learn more about the [compatibility levels supported](#).

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Understand Azure preview features

Azure is constantly evolving, and new features are released continuously. It may take some time before new features are generally available through the preview process. While preview features can benefit your applications, there are some tradeoffs around supportability.

Azure offers preview features that are designed for non-production usage. It's important to note that previews are subjected to reduced or different service terms. Depending on the region your resources are hosted, preview features may not be available and may have limited service level agreements (SLA), and functionality.

Microsoft doesn't recommend the use of preview features in a production environment, unless you're working directly with the product team to ensure support.

Preview features are broken down into the following categories:

- Private preview
- Public preview

Private preview features will require that Microsoft add your subscription to an allowlist for a given feature.

Public preview features are opted into in the portal but are available to everyone. Some features may require further opt-in at the individual resource. The public preview experience isn't consistent across Azure services.

You can check the latest public preview features by opening the [Azure portal preview page](#).

After public preview, the status of the feature changes to generally availability. General availability (GA) is the final release status, and it means the functionality is complete and accessible to all users.

Describe Azure database migration options

Many organizations are migrating their database platform to Azure SQL to reduce licensing costs. Migrating to Azure SQL platform is made easier by the Azure Database Migration Service (DMS). DMS supports both homogenous (for example, MySQL in a Virtual Machine to Azure SQL Database) and heterogenous sources (for example, Oracle in a Virtual Machine to Azure Database for PostgreSQL) migrations.

There are several tools available to help with the migration process. This next section looks at some of the options and methods for migration.

Azure Migrate tool

This migration tool provides a centralized location to assess and migrate on-premises servers, infrastructure, applications, and data to Azure. It will provide discoverability and proper assessments of your servers regardless of whether they are physical or VMWare/Hyper-V virtual machines.

Azure Migrate will also help to ensure that you select the appropriate size of virtual machine so that workloads will have enough resources available. In addition, the tool will provide a cost estimation so that you can budget accordingly.

In order to utilize the Azure Migrate tool, you must deploy a light-weight appliance, which can be deployed on a virtual or physical machine. Once the on-premises servers are discovered, the appliance will continually send metadata about each server (along with performance metrics) to Azure Migrate, which resides in the cloud.

The screenshot shows the Azure Migrate portal interface. On the left is a navigation sidebar with links like Overview, Migration goals, Servers, Databases, Data Box, Manage, Discovered items, Support + troubleshooting, and New support request. The main area has a heading 'Migrate your on-premises datacenter to Azure'. It features two large cards: 'Discover, assess and migrate servers' (with a red box around it) and 'Discover, assess and migrate databases'. To the right are sections for 'Assess and migrate web apps to Azure' and 'Migrate on-premises data to Azure'.

As shown above, the Azure Migrate experience can be kicked off from the portal to begin your migration process. The service consists of a unified migration platform, which provides a single portal to track your entire migration to Azure.

There are several other tools you can use to map your server estate and identify compatibility with your target Azure platform:

- **MAP Toolkit**—The Microsoft Assessment and Planning Toolkit automatically collects and provide a report containing the inventory of all SQL Servers in your network, version, and server information.

- Database Experimentation Assistant—This tool can be used to evaluate version upgrades of SQL Server by checking syntax compatibility and provides a platform to evaluate query performance on the target version.

Data Migration Assistant

The MAP toolkit and Database Experimentation assistant can help you identify your databases and highlight any incompatibilities or potential performance issues in your database, but the **Data Migration Assistant (DMA)** is a comprehensive toolkit that assesses, identifies new features you can use to benefit your application, and ultimately performs the migration. This tool can be used to migrate between versions of SQL Server, from on-premises to an Azure Virtual Machine or Azure SQL Database.

One of the main benefits of the DMA is the ability to assess queries both from Extended Event trace files and SQL queries from an external application, for example T-SQL queries in the C# application code for your application. You can generate a full report using a C# source and upload the migration assessment to the DMA. **The DMA mitigates the risk of moving to a newer version of SQL Server or to Azure SQL Database.**

Azure Database Migration Service

The Azure Database Migration Service is designed to support a wide mix of different migration scenarios with different source and target databases, and both offline (one-time) and online (continuous data sync) migration scenarios.

For online migrations, Azure Database Migration Service provides a highly resilient and self-healing migration service with reliable outcomes and near-zero downtime. Below are highlighted the main steps involved:

1. Fully load your on-premises database to Azure Database.
2. Continuously syncs new database transactions to the Azure target.
3. Cut over to the target Azure service when prepared. **You can stop the replication, and switch the connection strings in your application to the Azure Database.**

The offline source and target pairs are shown in Table 2 below:

Target	Source
Azure SQL Database	SQL Server
	RDS SQL
	Oracle
Azure SQL Managed Instance	SQL Server
	RDS SQL
	Oracle
Azure SQL Virtual Machine	SQL Server
	Oracle

Target	Source
Azure Cosmos Database	MongoDB
Azure Database for MySQL	MySQL
	RDS MySQL
Azure Database for PostgreSQL	PostgreSQL
	RDS PostgreSQL

The source and target pairs for online migration are shown in Table 3 below:

Target	Source
Azure SQL Database	SQL Server
	RDS SQL
	Oracle
Azure SQL Managed Instance	SQL Server
	RDS SQL
	Oracle
Azure SQL Virtual Machine	SQL Server
	Oracle
Azure Cosmos Database	MongoDB
Azure Database for MySQL	MySQL
	RDS MySQL
Azure Database for PostgreSQL	PostgreSQL
	RDS PostgreSQL
	Oracle

The Data Migration Service has a few prerequisites that are common across migration scenarios. You need to create a virtual network in Azure, and if your migration scenarios involve on-premises resources, you will need to create a VPN or ExpressRoute connection from your office to Azure. There are a number of network ports that are required for connectivity. Once the prerequisites are in place, the time to complete migration will depend on the data volume and rate of change in the databases in question.

There are a number of traditional, more manual approaches to migrating databases to Azure including backup and restore, log shipping, replication, and adding an Availability Group replica in Azure. These solutions were not designed primarily for performing migrations, but they can be used for that purpose. The technique you use for physically migrating your data will depend on the amount of downtime you can sustain during the migration process.

Learn more about the tools used to migrate SQL databases to Azure.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which tool would you use to assess and migrate your databases from an on-premises SQL Server to an Azure Virtual Machine?

- Microsoft Assessment and Planning Toolkit
- Database Experimentation Assistant
- Data Migration Assistant

Check Answers

Multiple choice

What does Microsoft guarantee if you upgrade versions of SQL Server but maintain the same compatibility level, on similar hardware?

- Syntax compatibility
- Execution Plan Shape
- Elapsed time of queries

Check Answers

Multiple choice

What is a valid reason for migrating your database to Azure SQL Database?

- Your applications need to run older versions of SQL Server, such as SQL Server 2016
- You want to eliminate the complexity of configuring and managing high availability, backups, and other database tasks
- You want to improve query performance of your application

Check Answers

Summary

In this module, you learned several migration options and usage to implement a successful database migration strategy. Understanding how compatibility level works is key to migrate your legacy databases to cloud successfully.

Now that you've reviewed this module, you should be able to:

- Understand how compatibility level affects query optimizer behaviors.
- Understand how Azure preview feature process works.
- Describe the various options when you are performing a migration.

Introduction

Azure SQL Database provides an industry-leading Platform-as-a-Service (PaaS) implementation of Microsoft SQL Server in the Azure cloud. If you want to migrate from an on-premises database to Azure SQL Database, you must plan carefully to ensure that the project runs smoothly.

Suppose you work for a company that builds bikes and bicycle parts. You have several legacy database servers that you want to upgrade, including a product database, a parts stock database, and a human resources database. **You also want to move from a capital expenditure model to an operational expenditure model, and benefit from the scalability and availability of the cloud.** You plan to migrate your servers to Azure SQL Database. Your board of directors has asked you to plan the migration project and has made you responsible for the smooth delivery of the working Azure SQL Database system.

In this module, you'll look at how to migrate SQL Server workloads to Azure SQL Database. You'll begin by exploring the pre-migration considerations you need to take into account before a migration, and how to create an Azure SQL Database. You'll then explore the different methods for offline and online migrations, and look at how to ensure service continuity of Azure SQL Database.

By the end of this module, you'll be able to plan a migration to Azure SQL Database confidently and run a migration project smoothly.

Learning objectives

In this module, you'll learn:

- **The considerations of migrating to Azure SQL Database**
- **The options to migrate to an Azure SQL Database offline**
- **The options to migrate to an Azure SQL Database online**
- **Service continuity for Azure SQL Database**

Course prerequisites

Students who take this training should have professional experience with SQL Server, and technical knowledge equivalent to the following courses:

- **Azure Fundamentals part 1: Describe core Azure concepts**

Choose the right SQL Server Instance option in Azure

Azure SQL Database is the collective term for Microsoft's SQL Server Platform-as-a-Service (PaaS) offering. It's fully managed and gives organizations a highly performing, reliable, and secure, general purpose relational database engine in the cloud.

In your bicycle manufacturing company, you've already identified and profiled the databases that you want to migrate to Azure SQL Database. Now, you want to plan the migration, considering data recoverability, disaster recovery, security, and other implementation details. You'd like to know if Microsoft provide tools to help with this process.

In this module, **the focus is on single databases and elastic pools**. A single database has its own resources and is deployed to a logical SQL Database server where it's managed. Elastic pool databases are deployed onto a single SQL Database server where resources are shared between all databases.

Benefits of Azure SQL Database (single and elastic pools)

The following sections categorize and summarize the benefits of deploying single and elastic pool databases:

Backup and recovery

- Automatic backup
- Point-in-time restore
- Backup retention 7 days+
- Long-Term Backup Retention stores backups for up to 10 years

High availability

- 99.99% availability guarantee
- Built-in availability with three secondary replicas
- Zone redundancy via Azure availability zones

Disaster recovery

- Geo-restore of database backups
- Active-geo replication between Azure regions

Service scalability

- Dynamic scale-up and scale-down
- Scale out with multiple shards
- Share compute resources between databases using elastic pools

Security

- Support for Azure Active Directory authentication
- Cloud-only security features such as Advanced Threat Protection
- **Transparent Data Encryption (TDE) enabled by default**
- **Support for dynamic and static data masking, row-level security, and Always Encrypted**
- Firewall allow list

Licensing

- DTU purchasing model for predictive costing
- vCore purchasing model, enabling storage to be scaled independently of compute
- Combine the vCore purchasing model with Azure Hybrid Benefit for SQL Server to realize cost savings of up to 30 percent

Tools to support your migration planning to Azure SQL Database

The following tools should be used in the discovery, planning, and assessment stage of your migration to Azure SQL Database:

1. **Microsoft Planning and Assessment.** Use this tool in the discovery stage to confirm the source environment that you're migrating from.
2. **Azure Database Migrate Service.** The Azure Database Migrate Service enables you to do large-scale database migration from within the Azure portal.
3. **Data Migration Assistant.** Use the Data Migration Assistant in the planning and assessment stage of a Data Platform Modernization project to check for compatibility issues that impact database functionality in Azure SQL Database.
4. **Database Experimentation Assistant.** If you're concerned about workloads that operate on the target server, use the Database Experimentation Assistant to assess if your target server can handle it.

Migration planning for Azure SQL Database

When you migrate to an Azure SQL Database, it's important to recognize that this product, although similar to SQL Server, is a distinctly separate application. There are considerations that need to be planned for before migration.

SQL agent jobs

Azure SQL Database doesn't provide functionality to host SQL Server agent jobs that must be migrated to complementary technologies such as:

- Azure Automation
- Elastic Database jobs

Security settings

Authentication with security principals in Azure SQL Database supports two types:

- SQL authentication
- Azure Active Directory authentication

Read scale-out

Read scale-out allows for read-only SQL workloads to be serviced by a secondary replica.

Retry application connections

When you access any cloud service, it's important to understand the requirements for retry logic so an application can recover from temporary issues and problems.

Retry Service Specific

Create an Azure SQL Database

To create an Azure SQL Database using the Azure portal, complete the following steps:

1. Sign in to the [Azure portal](#) with your subscription.
2. Select **Create a resource** in the upper-left corner of the Azure portal.
3. Locate the managed instance, and then select **SQL Database**.
4. Select **Create**.
5. On the **Basics** tab, in the **Project Details** section, type or select the following values:

Property	Value
Subscription	Select the correct subscription
Resource group	Select Create new , type cto , and select OK

6. In the **Database Details** section, type or select the following values:

Property	Value
Database name	mySampleDatabase
Server	Select Create new
Server name	mysqlserver , along with some numbers for uniqueness.
Server admin login	azureuser
Password	Type a complex password that meets password requirements.
Location	Choose a location from the drop-down, such as West US 2.

7. In the “Want to use SQL elastic pool” section, select the **No** option.

8. In **Compute + storage**, select **Configure database** and, for this quick start, select **vCore-based** purchasing options.
9. Select **Serverless**.
 - Review the settings for **Max vCores**, **Min vCores**, **Autopause delay**, and **Data max size**, and change them if necessary - Accept the preview terms and click **OK** - Select **Apply**
10. Select the **Additional settings** tab.
11. In the **Data source** section, under **Use existing data**, select **None**.
12. Leave the rest of the values as default and select **Review + Create** at the bottom of the form.
13. Review the final settings and select **Create**.
14. On the **SQL Database** form, select **Create** to deploy and provision the resource group, server, and database.

Migrate SQL Server to Azure SQL Database offline

If you afford to take the database offline while you migrate to Azure, then you have a choice of techniques **you can use to migrate the schema and data.**

In your bicycle manufacturing company, the HR database is considered business-critical but is rarely used at weekends. You've planned to execute an offline migration between Friday evening and Monday morning, but you want to assess the best migration method.

In this unit, the focus is on the tools and methods for migrating SQL Server databases to Azure SQL Database offline.

NOTE: It's assumed that all pre-migration checks have been done with the Data Migration Assistant and the Database Experimentation Assistant. This process ensures that feature and compatibility issues are addressed, and workloads have been simulated.

Migrate using the Data Migration Assistant

You use the Data Migration Assistant to help migrate your SQL Server workload to a single or a pooled Azure SQL Database, if your organization can tolerate downtime. Here are the five steps required to do this work:

1. Use the Data Migration Assistant to assess the database for compatibility issues.
2. Use the compatibility report to prepare the fixes required in a Transact SQL script.
3. Make a copy of the database that's transactionally consistent.
4. Deploy the Transact SQL script with the fixes to the copy of the database.
5. Migrate the database copy to a new Azure SQL Database by using the Data Migration Assistant.

During migration, there are several best practices you can employ for importing the database into Azure SQL Database, including:

- Choose the highest service tier and compute size that your budget allows to maximize the transfer performance. To save on cost, you can scale down after the migration completes.
- Minimize the distance between your BACPAC file and the destination data center.
- Disable autostatistics during migration.
- Partition tables and indexes.
- Drop indexed views and recreate them when finished.
- Remove rarely queried historical data to another database and migrate it to a separate Azure SQL Database. You can then query this historical data using elastic queries.
- After migration, update of all the statistics in the database.

Migrate to Azure SQL Database using BACPAC

You can import a SQL Server database into an Azure SQL Database using a BACPAC file. Import the data from a BACPAC file stored in Azure Blob storage (standard storage only) or from local storage in an on-premises location.

To maximize import speed by providing more and faster resources, scale your database to a higher service tier and compute size during the import process. You can scale down after the import is successful.

Import from a BACPAC file in the Azure portal

The Azure portal only supports creating a single database in Azure SQL Database, and only from a BACPAC file in Azure Blob storage.

1. To import from a BACPAC file into a new single database using the Azure portal, open the appropriate database server page and then, on the toolbar, select **Import database**.
2. Select the storage account and container for the BACPAC file, and then select the BACPAC file from which to import.
3. Specify the new database size (usually the same as the origin) and provide the destination SQL Server credentials.
4. Select **OK**.
5. To monitor an import's progress, open the database server page and, under **Settings**, select **Import/Export history**. When successful, the import has a **Completed** status.

You could also use SqlPackage to import a BACPAC file as it's more performant than using the Azure portal method. The following SqlPackage command imports the AdventureWorks2008R2 database from local storage to an Azure SQL Database server called mynewserver20170403. It creates a new database called myMigratedDatabase with a Premium service tier and a P6 Service Objective. Change these values as appropriate for your environment.

```
SqlPackage.exe /a:import /tcs:"Data Source=mynewserver20170403.database.windows.net;
```

Streamline migrations with the Azure Database Migration Service

The Azure Database Migration Service (DMS) is fully managed and 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. DMS is a free service that supports migrations of different databases to Azure database offerings. DMS can migrate MySQL, PostgreSQL, and MariaDB databases to Azure Database for MySQL/PostgreSQL/MariaDB, and it also supports SQL Server migrations, including SQL MI.

The service uses the Data Migration Assistant to generate assessment reports that provide recommendations to guide you through the changes needed before a migration. You do any remediation that's required. When you're ready to begin migration, Azure Database Migration Service does all of the required steps. You can fire and forget your migration projects, safe in the knowledge that the process uses best practices determined by Microsoft.

Before using the Data Migration Assistant, you must register a resource provider in Azure, and create an Azure Database Migration Service instance.

Migrate using the Azure Database Migration Service

To use the Azure Database Migration Service to migrate a database to Azure SQL Database, complete the following steps:

1. Create a migration project
2. Specify source details
3. Specify target details
4. Select source databases
5. Configure migration settings
6. Review the migration summary
7. Run and monitor the migration
8. Complete migration cutover

After the full database backup is restored on the target instance of Azure SQL Database, the database is available to do a migration cutover.

Migrate SQL Server to Azure SQL Database online

If you need a database to remain available to users throughout the migration process, you can use transactional replication to move the data.

In your bicycle manufacturer, warehouses run on a 24 hour, 7 days a week basis and there are no periods of inactivity. Your board of directors wants to be sure that the part stock database is constantly available, even during the migration to Azure SQL Database.

Here, you'll learn how to use transactional replication to perform an online database migration.

What is transactional replication?

Where the source system is business-critical and must remain online throughout the migration, transactional replication uses an initial snapshot to copy the data to the Azure SQL Database. The source and target systems are kept in sync until the final cutover takes place. Configuration of transactional replication is done through SQL Server Management Studio, or by executing Transact-SQL statements on the publisher. Transactional replication requires the following components:

Publisher: a database instance that hosts the data to be replicated (source).

Subscriber(s): a database instance that receives the data being replicated by the *Publisher* (target(s)).

Distributor: a database instance that stores the replication changes made at the *Publisher* (source) that are required at the *Subscriber(s)* (target(s)).

Article: a database object; for example, a table that's included in the *Publication*.

Publication: a collection of one or more articles from the database being replicated.

Subscription: a request from a *Subscriber* for a *Publication* from a *Publisher*.

The publisher and distributor must be at least the following version and update:

- 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) CU8 or SP3

Transactional replication considerations

- Replicated tables must have a primary key
- Transactional replication can't be configured from the Azure portal
- Using the latest version of SQL Server management tools is recommended

Here are the roles that can be used with Azure SQL Database:

Role	Single and pooled databases	Managed instance databases
Publisher	No	Yes
Distributor	No	Yes
Pull subscriber	No	Yes
Push subscriber	Yes	Yes

When synchronization is complete and you're ready to migrate, change the connection string of your applications to point them to your Azure SQL Database. After transactional replication drains any changes left on your source database, and all your applications point to Azure SQL Database, you can uninstall transactional replication. Your Azure SQL Database is now your production system.

NOTE: A pull subscription isn't supported when the distributor is a managed instance database and the subscriber is not.

Load and move data to Azure SQL Database

After you complete the migration of data to Azure SQL Database, it's important to assess and optimize the configuration of your new database to give the best service to your users.

You've completed the migration of all databases to Azure for your bicycle manufacturer. Now, you want to make sure that the new databases are resilient, robust, and able to handle the load that you expect users to place on them.

In this unit, you'll explore the post migration steps required to ensure service continuity of your database in backup, high availability, disaster recovery, and scalability.

Define backup and recovery options for Azure SQL Database

In Azure SQL Database, backups are taken automatically and kept for between 7 and 35 days. With the DTU model, a database under the Basic tier benefits from seven-day retention. Standard and Premium databases have 35 days retention. Under the vCore purchasing model, the default retention period is seven days, which can be increased up to 35 days.

Backups are stored on Azure read-access geo-redundant storage (RA-GRS) to ensure they're still available during a data center outage. To provide a point-in-time restore (PITR) capability, Azure SQL Database does transaction log backups every 5-10 minutes. There are differential backups every 12 hours. Alongside PITR, geo-restore is used to restore databases to another geographical region during a region-wide outage.

Backups are kept for longer than 35 days by using the Long-Term Retention capability. You can keep weekly, monthly, or full backups in this capability for up to 10 years in RA-GRS storage containers. This backup retention is ideal for organizations that have to meet legislative or regulatory requirements.

Define high availability options for Azure SQL Database

Azure SQL Database high availability is provided out of the box to guarantee that your database keeps running 99.99% of the time. You won't have the worry of maintenance operations and outages. When the underlying SQL instance fails over, downtime isn't noticeable providing you employ retry logic within your application.

There's a choice of two high-availability architectural models used in Azure SQL Database, depending on the service tier you've selected:

Basic, Standard, and General Purpose service tier availability

On this tier, high availability is achieved by using the high availability and reliability of the remote premium storage tier. You can transfer the compute from the active node to other nodes.

Premium and Business Critical service tier availability

High availability on this tier is achieved by using a technology similar to Always On availability groups. Both compute and storage are replicated to additional nodes.

Define disaster recovery options for Azure SQL Database

Azure SQL Database provides the following business continuity capabilities:

- **Active geo-replication:** provides a readable secondary replica for a given database in the same or different Azure region. Active-geo replication asynchronously replicates data by using the same technology as Always On availability groups. Up to four secondary replicas are permitted, and can be used for read-only workloads. You invoke the failover process using the application or via manual procedures with the Azure CLI, PowerShell, Transact-SQL, or the REST API.
- **Auto failover groups:** builds on the capabilities of active geo-replication by replicating and failing over a group of databases on an Azure SQL Database server. Grouping the databases allows multiple databases to be recovered if there's an outage.
- **Geo-restore:** uses geo-redundant backups to recover a database to any Azure SQL Database server, in any region. Backups are stored on geo-replicated storage, which means there's a delay between when the backup is taken, and when it's replicated to the other region.
- **Zone-redundant databases:** by default, replicas in the premium availability model are located in the same physical data center. Azure availability zones allow different replicas to be hosted in different zones (data centers) within the same region.

Define service scalability options for Azure SQL Database

Azure SQL Database supports vertical scaling, known as scale-up, and scale-down, and horizontal scaling (scaling out), known as sharding (an architectural design pattern).

Single and elastic pool databases can be scaled up and down to accommodate increases in application workload. Scaling might occur at a particular point in the day, month, or year when the workload peaks or troughs. When you scale a database up or down, the performance objective of the database will increase or decrease.

Single and elastic pool databases can be sharded. **Sharding is where data is distributed across multiple databases, known as shards.** Each shard is an individual database that contains data relevant to the shard. Relevance is decided by the sharding key, which is used to distribute the data via data-dependent routing. This sharding is useful when parts of the data are only available in different regions, or where the connections should be load balanced.

Vertical scaling can be done via the Azure portal, PowerShell, T-SQL, Azure CLI, or the REST API. Horizontal scaling is done using the Elastic Database Client Library. You can use the Elastic Database Split-Merge tool to split and merge sharded databases.

Summary

In your bike manufacturing company, you had several legacy database servers that you needed to upgrade. You also wanted to move from a capital expenditure model to an operational expenditure model. By moving to Azure SQL Database the organization benefits from the scalability and availability of the cloud.

You created and executed a plan to migrate your servers to Azure SQL Database using both offline and online methods. Now, with the migration complete, your users have the best availability and scalability for their data and you're confident that you can respond to future changes in demand quickly.

Learn more

- [Azure Database Migration Guide](#)
- [Transactional Replication with Azure SQL Database](#)

Introduction

Azure SQL Database managed instances make it easy and quick to lift and shift a database from on-premises into the cloud.

Suppose you work for a sports clothing retail company. Your organization has been moving many of its systems into Azure to realize improved cost-of-ownership and better availability. So far, you haven't considered migrating your business-critical databases to Azure because you've heard that databases often need modification to work in Azure SQL Database.

You want to evaluate Azure's database hosting solutions, choose the best one to host your database, and do the migration. You'd prefer to migrate with as little database administrator and developer time as possible to keep the project budget low.

In this module, you'll learn about Azure SQL Database managed instance, how it differs from other database solutions available in Azure, and how to choose the best cloud database solution for your project.

By the end of this module, you'll know how to plan and execute a database migration project to Azure SQL Database managed instance.

Learning objectives

By the end of this module, you'll see how to:

- Evaluate migration scenarios to Azure SQL Database managed instance
- Migrate to Azure SQL Database Managed Instance
- Load and move data to Azure SQL Database managed instance

Prerequisites

- Understand SQL Server Database file administration
- Successfully log into the Azure portal
- Understand the storage options in Azure
- Understand the Azure compute options

Evaluate migration scenarios to SQL Database Managed Instance

Azure SQL Database managed instance is designed to make it easy to host existing databases in the cloud by providing almost 100 percent compatibility with on-premises versions of SQL Server.

In your sports clothing company, you have a database that stores the product details for your entire catalog. The website uses the database to display product details to customers, by the sales representatives' smartphone apps to keep them informed about the catalog, and by a data analysis solution to populate product dimensions in a data cube. The database is considered business-critical by the board of directors. You've been asked to migrate this database to the cloud so the systems that depend on it need as little modification as possible. You want to evaluate Azure SQL Database managed instance for this project.

Here, you'll learn about the key features of managed instances, and how they make database migration easy. You'll also learn how to create a managed instance in your Azure subscription.

What is Azure SQL Database managed instance?

Azure SQL Database managed instance is the newest SQL-based platform as a service (PaaS) capability that's part of the Azure SQL Database. The Azure SQL Database family includes the single Azure SQL Database, Azure SQL Database elastic pools, and Azure SQL Database managed instance. The goal of Azure SQL managed instance is to provide SQL Server applications with a fully managed PaaS experience in the Azure cloud. Azure SQL Database managed instance is also a fully featured SQL PaaS experience with all the critical features and capabilities of the SQL Server platform.

Azure SQL Database managed instance is designed to enable a "lift and shift" solution for customers. The managed instance looks to bring applications, databases, and supporting technologies to Azure PaaS. Previously, without Azure SQL Database managed instance, migration scenarios where an organization's application required access to any technology outside of the database (for example SQL Agent jobs, cross database joins, and SQL Server Integration Services) would be prevented from moving to the cloud. The only way a DBA or developer could migrate an on-premises application would be to employ one of the following approaches:

- Move the database and supporting technologies to an infrastructure as a service (IaaS) model.
- Rewrite the application with a fully PaaS model on Azure SQL Database, with additional development to address migration blockers.

The discussion usually concerns whether your organization has the resources and capabilities to rewrite the application to fit into a PaaS model on the Azure SQL Database. The organization would also need to control the application code as vendors will often not support any changes made to their application. The result is that most organizations looking to migrate their applications to Azure choose SQL Server on IaaS. You're likely to get the full SQL Server experience in Azure while avoiding the overhead of rewriting current applications.

While Azure SQL Database is powerful, most current applications have too many technologies that exist outside the database scope. This situation leads to vendor support challenges and expensive development cycles that most organizations can't support. Azure SQL Database managed instance, code-named "Cloud Lifter", was designed to eliminate these blockers to migrating your application databases to a SQL-based PaaS solution in Azure, without having to redesign the application.

Key features

The most important features of Azure SQL Database managed instance include:

- **Backwards compatibility.** Managed instance provides backward compatibility to SQL Server 2008 databases. Direct migration from SQL Server 2005 database servers is also supported, with the compatibility level for migrated SQL Server 2005 databases being updated to SQL Server 2008. Azure SQL Database managed instance also provides access to newer technologies. You can use database compatibility level 150, which enables many additional capabilities.
- **Easy lift and shift.** Managed instance has close to 100 percent compatibility to SQL Server. This compatibility includes core SQL Server components, programmability enhancements, instance-scoped features, such as cross database joins, and management tools that most existing SQL-based applications need to function correctly. Ultimately, SQL Database managed instance enables database migration to a fully managed database service, without redesigning the application.

Easy migration: nearly 100% like SQL Server

Data migration <ul style="list-style-type: none"> • Native backup/restore • Configurable DB file layout • DMS (migrations at scale) 	Security <ul style="list-style-type: none"> • Integrated Auth (Azure AD) • Encryption (TDE, AE) • SQL Audit • Row-Level Security • Dynamic Data Masking
Programmability <ul style="list-style-type: none"> • Global temp tables • Cross-database queries and transactions • Linked servers • CLR modules 	Operational <ul style="list-style-type: none"> • DMVs & XEvents • Query Store • SQL Agent • DB Mail (external SMTP)
	Scenario enablers <ul style="list-style-type: none"> • Service Broker • Change Data Capture • Transactional Replication

- **Fully managed PaaS.** PaaS benefits include removing the need for managing hardware and all the overhead that comes from doing physical maintenance on SQL Server machines. You also have the benefits of quickly scaling up and scaling down, and provisioning resources in the cloud. The SQL Server operating system, commonly known as the SQLOS, is maintained by PaaS with automated patching and version upgrades.
- **Security features.** You can enable Advanced Data Security features at the SQL Database managed instance scope just as you do at the database level. These features include the Vulnerability Assessment and the Advance Threat Protection settings. You can also enable specific Advanced Threat Detection features and schedule periodic recurring scans that send security reports to administrators. Finally, at the managed instance level, you can configure Transparent Data Encryption (TDE) and whether you want to bring your own key (BYOK) for encryption.
- **Secure network isolation.** One of the unique aspects of managed instance, network security isolation is where managed instance has complete security isolation from any other tenant in the Azure cloud. In a typical default deployment SQL endpoint, managed instance is solely exposed through a private IP address that only allows connectivity from private Azure networks or hybrid networks. For on-premises applications to connect to managed instance, you'd need an Azure ExpressRoute configuration or a VPN gateway.
- **Instance failover groups.** An instance failover group is a set of databases managed by a single database server, or within a single managed instance, that can fail over as a unit to another region. You use instance failover groups when all or some of the primary databases have gone offline due to an outage in the primary region.

Migration tools

There are several tools you can use to make the discovery, planning, and assessment stage of your migration to SQL Database managed instance easier:

- **Microsoft Planning and Assessment.** Use this tool in the discovery stage to confirm the source environment that you're migrating from. Microsoft Planning and Assessment helps you understand the configuration of your source SQL Server, how many instances are installed, and the components running on each instance. You can also use the tool to confirm the version and configuration of the Windows Server that it's running on.
- **Azure Database Migrate Service.** The Azure Database Migrate Service helps you do large-scale database migration from within the Azure portal. Azure Database Migration Service integrates some of the functionality of existing tools and services to give customers a comprehensive, highly available solution. The tools include:
 - Database Migration Assistant
 - SQL Server Migration Assistant
 - Data Experimentation Assistant
- **Data Migration Assistant.** Use the Data Migration Assistant in the planning and assessment stage of a data platform modernization project. You can check for compatibility issues that impact database functionality in SQL Database managed instance. Also, you use Data Migration Assistant to review performance and reliability improvements for a target environment before you do the migration. You can then incorporate these improvements into your plan.
- **Database Experimentation Assistant.** If you're concerned about the workloads that will operate on the target server, use the Database Experimentation Assistant to assess if your target server can handle the workload. Customers who upgrade from earlier SQL Server versions to a more recent version of SQL Server on Azure virtual machines, can use the analysis metrics to give comparison data. This data helps you make decisions on whether the targeted version would provide a better experience after a migration.

Migration planning

A database migration project is often involved and complex. Many things can go wrong and prevent you completing the project carefully, affecting users. To reduce the chances of such problems arising, consider the following issues before you migrate.

Licensing

Six types of subscription models are used for Azure SQL Database managed instance. Assess which one is the most appropriate for your migration scenario:

- Enterprise Agreement (EA)
- Pay-as-you-go
- Cloud Service Provider (CSP)
- Enterprise Dev/Test
- Pay-as-you-go Dev/Test
- Subscriptions with monthly Azure credit for Visual Studio subscribers

Managed instance sizing

Two generations of Azure SQL Database managed instance have hardware limitations. Choose the version that will meet your needs.

	Gen4	Gen5
Hardware	Intel E5-2673 v3 (Haswell) 2.4-GHz processors, attached SSD vCore = 1 PP (physical core)	Intel E5-2673 v4 (Broadwell) 2.3-GHz processors, fast NVMe SSD, vCore=1 LP (hyper-thread)
vCores	up to 24 vCores	up to 80 vCores
Memory	7 GB per vCore	5.1 GB per vCore
Max OLTP memory	3 GB per vCore	2.5 GB per vCore
Max instance storage (General Purpose)	8 TB	8 TB
Max instance storage (Business Critical)	1 TB	1 TB, 2 TB, or 4 TB depending on the number of cores

You select the generation of the Azure SQL Database managed instance, and a service tier: General Purpose or Business Critical.

Database compatibility

To support older databases on new versions of the database engine, SQL Server uses database compatibility levels. Identify the latest supported compatibility level of the application that's using SQL Server. Match that to the compatibility level of the database that the application queries. SQL Server 2000 has a compatibility level of 80, and each next version of SQL Server has +10 compatibility by default, up to 150 for SQL Server 2019.

SQL Server 2017 supports compatibility levels from 100 (default for SQL Server 2008) to 140 (default for SQL Server 2017). If your application requires compatibility level 100, and you use SQL Server 2014, you can safely move to SQL Database managed instance. But if you use an application with compatibility level 80, running on SQL Server 2008, you can't move it to **SQL Database managed instance as it supports compatibility levels of 100 and newer**. In this situation, you'll have to migrate from compatibility level 80 to 100 before you can move the database to an Azure SQL Database managed instance.

Networking

Azure SQL Database managed instance must be deployed within an Azure virtual network, with the subnet dedicated to managed instances only. SQL Database managed instance is fully isolated. The compute and storage are placed in a virtual cluster that's fully isolated from all other tenants in Azure.

Connect your on-premises resources using VPN tunneling or a route gateway to SQL Database managed instances. You can then use these instances as any others in your network. By connecting on-premises resources to SQL Database managed instance like this, you use managed instance as an extension of your on-premises datacenter.

If you have any back-end or front-end subnets in your on-premises network, you can establish VNET-to-subnet connections between them. You then use SQL connections from your web applications or link Azure SQL managed instance to your on-premises database. SQL Database managed instance becomes just an extension to your on-premises SQL solution and your organization's on-premises network.

Application location

When you've migrated your SQL Server database to Azure SQL Database managed instance, you need to decide where to host the applications that work with the database. Leaving the applications on-premises, when the database is in Azure, will cause latency issues. Consider moving your applications onto Azure to keep them close to the database.

Automated backups

Check that the automated backup schedules work for your organization. SQL Database managed instance automatically creates database backups that are kept between 7 and 35 days. Azure read-access geo-redundant storage (RA-GRS) is used to make sure they're preserved, even if the datacenter is unavailable. This capability is built in as part of the Azure SQL Database experience. If you need backups to be available for an extended period (up to 10 years), you can configure long-term retention on a single database.

The screenshot shows the Azure portal interface for managing a database named 'tpcc5000'. On the left, there's a sidebar with various navigation options like Overview, Activity log, and Settings. The main area shows 'Essentials' information including Resource group, Status (Online), Location (Australia Southeast), Subscription name, and Subscription ID. A modal window titled 'Configure backup retention' is open, asking 'How long would you like backup to be kept?'. The dropdown menu shows '7' days selected. At the bottom right of the modal is an 'OK' button.

SQL Database creates full backups every week, differential backups every 12 hours, and transaction log backups every 5-10 minutes. The backups are stored in RA-GRS storage blobs that are replicated to a paired datacenter for protection against an outage. When you restore a database, the service figures out which full, differential, and transaction log backups need to be restored.

COPY_ONLY backup is the only manual method that's allowed. The transaction log must be preserved for automated backup operations and point-in-time restore operations in managed instance.

Create an Azure SQL Database managed instance

Create an Azure SQL Database managed instance by following these steps:

1. Sign in to the [Azure portal](#), and then select **Create a resource** in the upper-left corner.
2. Search for **managed instance**, select **Azure SQL managed instance**, and then select **Create**.

The screenshot shows the Azure portal's 'Create an Azure SQL Managed Instance' page. At the top, there's a 'Report a bug' button and a search bar. Below that, the URL is 'Home > New > Azure SQL Managed Instance'. The main heading is 'Azure SQL Managed Instance' with a Microsoft logo. A large blue 'Create' button is prominently displayed. To the left of the button is a blue icon depicting a server tower and a cloud. Below the button, there's a detailed description of what a Managed Instance is: 'SQL Database Managed Instance is a deployment option in Azure SQL Database that is highly compatible with SQL Server, providing out-of-the-box support for most SQL Server features and accompanying tools and services. Managed Instance also provides native virtual network (VNET) support for an isolated, highly-secure environment to run your applications. Now you can combine the rich SQL Server programming surface area in the cloud with the operational and financial benefits of an intelligent, fully-managed database service, making Managed Instance the best PaaS destination for your SQL Server workloads.' Below this description is a 'Useful Links' section with links to Documentation, Pricing details, Migrate with Azure Database Migration Service, and Deploy Managed Instance inside a new virtual network.

3. Fill out the SQL managed instance form using the information in the following table:

Setting	Suggested value	
Subscription	Your subscription.	
managed instance name	Any valid name.	
managed instance admin login	Any valid username. Don't use "serveradmin" because that's a reserved server-level role.	
Password	Any password longer than 16 characters and meeting the complexity requirements.	
Time zone	The time zone to be observed by your managed instance.	
Collation	The collation you want to use for the managed instance. If you migrate databases from SQL Server, check the source collation by using <code>SELECT SERVERPROPERTY('N'Collation')</code> and use that value.	
Location	The Azure region in which you want to create the managed instance.	
Virtual network	Select either "Create new virtual network" or a valid virtual network and subnet.	
Enable public endpoint	Check this option to enable a public endpoint, which then helps clients outside Azure to access the database.	
Allow access from	Select from Azure services, the internet, or no access.	
Connection type	Choose between a Proxy and a Redirect connection type.	
Resource group	A new or existing resource group.	

The screenshot shows the 'Configure performance' dialog box for creating a new SQL managed instance. The 'General Purpose' workload tier is selected, which includes 8 / 16 / 24 / 32 / 40 / 64 / 80 vCores, 32 GB - 8 TB storage capacity, and fast storage. The 'Business Critical' tier is also listed for IO-intensive workloads. Compute generation is set to Gen5. vCores are set to 8. Storage is set to 256 GB. Days of backup retention is set to 7 days. There is a 'Save money' checkbox where users can confirm they have a valid license. A note at the bottom states that the displayed price is an estimate per month.

4. Select **Pricing tier** to size compute and storage resources, and to review the pricing tier options.
5. When you're finished, select **Apply** to save your selection, and then select **Create** to deploy the managed instance.
6. Select the **Notifications icon** to view the status of the deployment.
7. Select **Deployment in progress** to open the managed instance window to further monitor the deployment progress.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

Load and Move data to SQL Database Managed Instance

There are several ways to check compatibility for your database and migrate the data. Make sure you understand these methods to choose the best one for your project.

In your sports clothing company, you've completed the evaluation and chosen to migrate the product database to Azure SQL Database managed instance. Next, you want to execute your migration project. In the first stage, you'll identify any compatibility issues that might cause problems if you don't address them. Then you want to migrate the data.

Here, you'll learn how to assess compatibility with managed instance for a production database. You'll also see how to choose and execute the right migration method.

Evaluating Azure SQL Database managed instance compatibility

You previously viewed the compatibility options between Azure SQL Database, SQL Database managed instance, and SQL Server virtual machines. In this unit, you'll focus on the migration process and methods targeted to SQL Database managed instance. Before you start moving databases from your on-premises SQL Server environment, it's important to address any compatibility concerns.

The managed instance deployment option is designed to provide an easy lift-and-shift migration experience for most applications that use SQL Server on-premises or on virtual machines. However, you might sometimes require features or capabilities that aren't yet supported and the workaround cost is high.

Features such as cross-database queries, cross-database transactions within the same instance, linked servers, CLR, global temp tables, instance level views, Service Broker and more aren't supported on Azure SQL Database. However, these features are supported with SQL Database managed instance. To help you assess compatibility, Microsoft has created some tools to evaluate Azure compatibility, and address migration challenges.

Data Migration Assistant

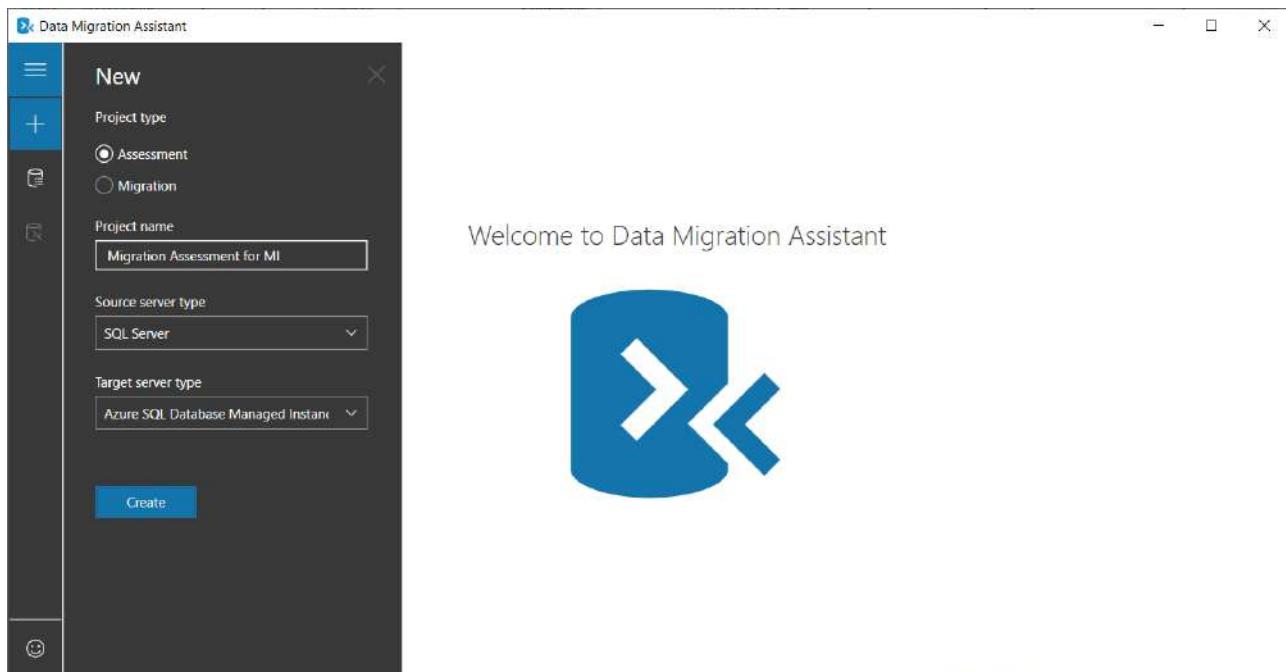
Use Data Migration Assistant (DMA) to detect potential compatibility issues that impact database functionality on Azure SQL Database, SQL Database managed instance, SQL Server, and SQL Server virtual machines. DMA provides the ability to do a migration assessment, and execute a migration project. From DMA 4.3, DMA supports managed instance as an assessment option, and a migration destination.

The result of a DMA project is a report of the equivalent features between your source and destination, and a list of any compatibility issues.

Running a Data Migration Assessment

In the following steps, you'll execute a DMA assessment, and examine the output of the scan:

1. Open **Data Migration Assistant**, and then create a new project.
2. Choose **Assessment** as the project type, and name it so you can go back and evaluate previous executions.
3. Select the source server type and the target server type. In this case, SQL Server is the source server type. The target server type is Azure SQL Database managed instance.
4. Click **Create**.



5. Select the report types. In this case, you'll use both **Check database compatibility** and **Check feature parity**.

← Migration Assessment for MI

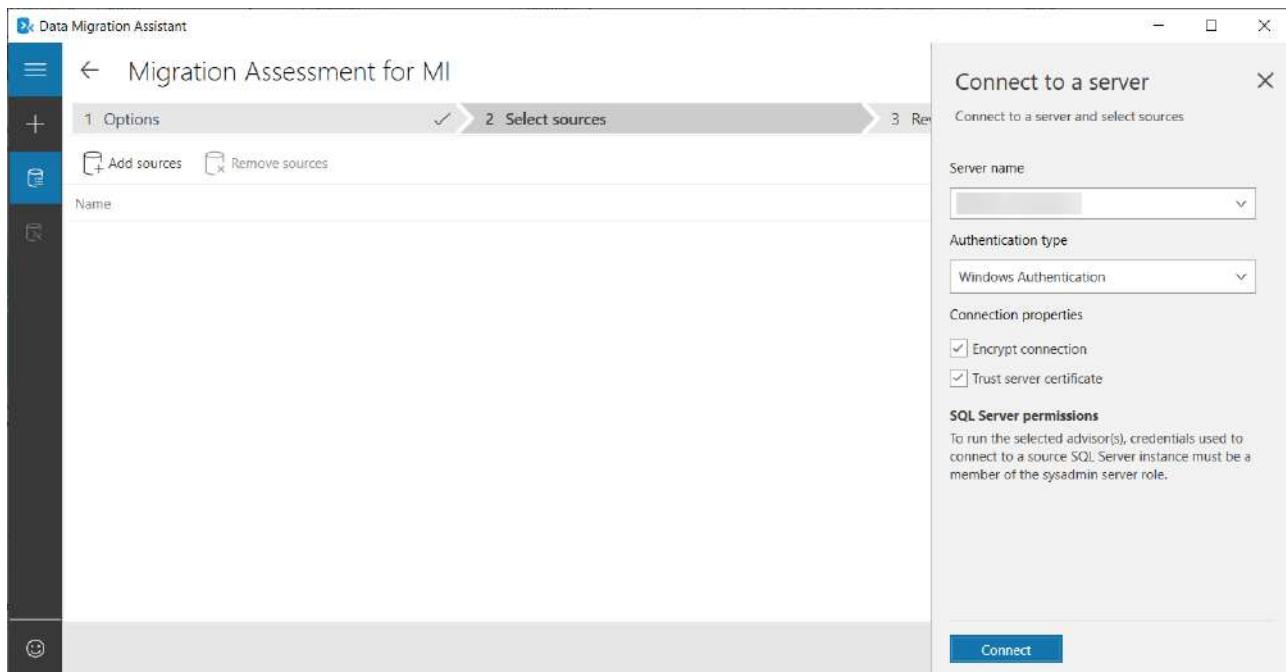
1 Options 2 Select sources 3 Review results

Select report type

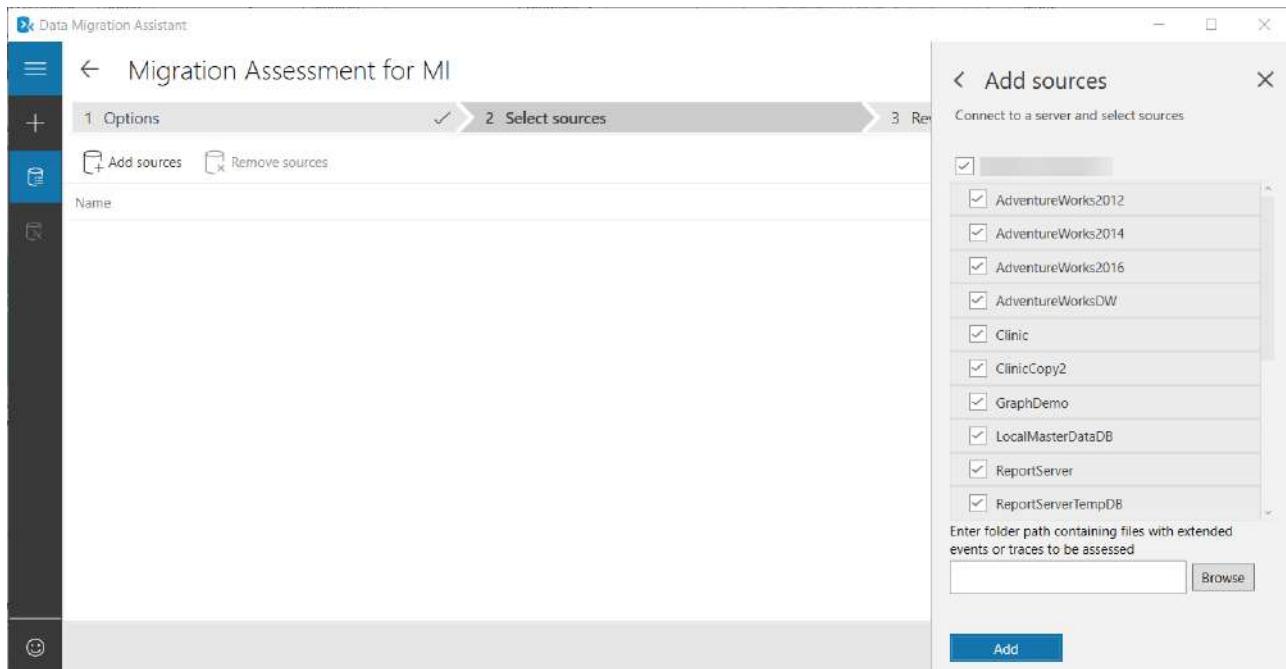
-  Check database compatibility
Discover migration blocking issues and deprecated features by analyzing databases you choose in your source server to be migrated to SQL Database.
-  Check feature parity
Discover unsupported or partially-supported features and functions that your applications may rely on. Get guidance around these areas that may need some re-engineering.
-  Benefit from new features (coming soon...)
Discover new SQL Database features that are applicable to the databases in your source once migrated to SQL database platform.

Next

6. Connect to your source system. In this case, a SQL Server 2017 environment has been chosen. You'll use the default authentication type. Click **Connect**.



7. Choose the databases you would like to analyze, and then click **Add**.



8. Click **Start Assessment**.

When you get the output, there's the option to examine SQL Server feature parity and the compatibility issues. First, you'll examine the SQL Server feature parity report.

Because you chose managed instance as the destination, you'll see a small list of features that don't have parity with SQL Server. In this example, the only reported issue is that PowerShell isn't a supported job-step type in Azure SQL Database managed instance.

Migration Assessment for MI

Target Platform: Azure SQL Database Managed Instance
MININT-SQLONAIR / SQL Server 2017

Feature parity (1)

Recommendation	Impacted objects
Unsupported features (1)	PowerShell job step is not supported in Azure SQL Database...

Impact: It is a job step that runs a PowerShell scripts. Use SQL Server Agent to run SQL Server PowerShell scripts at schedule times.

Recommendation: Find all agent jobs including PowerShell job step and remove the PowerShell job step from those jobs.

Object details:

- Type: Job step Name: syspolicy_purge_history.Erase Phantom System Health Records.
- This job step is not supported.

Select **Compatibility issues**. You now have a report of all the compatibility issues per database, and for each applicable database compatibility level.

Migration Assessment for MI

Target Platform: AdventureWorks2016 / SQL Server 2017 Compat 130 Size 20.07 GB

Compatibility 140 (2) **Compatibility 130 (2)**

Issue	Impacted objects
Unqualified Join(s) detected	Unqualified Join(s) detected
Full-Text Search has changed syntax	Full-Text Search has changed syntax

Impact: Starting with database compatibility level 90 and higher, in rare occasions, the 'unqualified join' syntax can cause 'missing join predicate' warnings, leading to long running queries.

Recommendation: An example of "Unqualified join" is select * from table1, table2 where table1.col1 = table2.col1

Object details:

- Type: Procedure Name: dbo.uspGetOrderTrackingBySalesOrderID
- Type: Procedure Name: dbo.uspGetOrderTrackingBySalesOrderID

You'll see the details and the affected objects. You can then open the supporting documentation and address the issues.

Migrate using backup and restore from the URL

If you can accept downtime, the restore of a database from a URL is a valid method for an offline migration. The restoration of native backups from SQL Server, uploaded to Azure Storage, to Azure SQL Database managed instance, enables quick and easy offline database migration.

Restore the database from a backup file

In SQL Server Management Studio (SSMS), follow these steps to restore a database to your managed instance. The database backup file is stored in an Azure Blob storage account.

1. Open SMSS and connect to your managed instance.
2. From the left menu, right-click your managed instance and select **New Query** to open a new query window.
3. Run the following T-SQL command, which uses a preconfigured storage account and key to create a credential in your managed instance.

```
CREATE CREDENTIAL [https://mtutorials.blob.core.windows.net/databases]
    WITH IDENTITY = 'SHARED ACCESS SIGNATURE',
    SECRET = 'add your shared access signature here'
```

4. To check your credential, run the following command, which uses a container URL to get a backup file list.

```
RESTORE FILELISTONLY FROM URL =
'https://mtutorials.blob.core.windows.net/databases/dbbu.bak'
```

5. Run the following command to restore the database:

```
RESTORE DATABASE [database name] FROM URL =
'https://mtutorials.blob.core.windows.net/databases/dbbu.bak'
```

6. Run the following script to track your restore's status:

```
SELECT session_id AS SPID, command,
    a.text AS Query, start_time, percent_complete,
    dateadd(second,estimated_completion_time/1000,
    getdate()) AS estimated_completion_time
FROM sys.dm_exec_requests r
CROSS APPLY sys.dm_exec_sql_text(r.sql_handle) a
WHERE r.command IN ('BACKUP DATABASE', 'RESTORE DATABASE')
```

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

Managing encrypted databases

Transparent Data Encryption (TDE) is a SQL Server technology that ensures databases are encrypted at rest. These databases can only be read when the certificate used to encrypt the data is used to decrypt the database and database backups at the destination. When you migrate a database protected by TDE to a managed instance using native restore, the certificate from the on-premises SQL Server must be migrated before database restore.

To address TDE enabled databases, you can either use Azure Database Migration Service (DMS) or manually decrypt the database backups using a certificate export tool and PowerShell.

Manually migrate the certificate

To migrate the certificate of TDE databases to Azure SQL DB MI, you'll need this software:

- **Pvk2Pfx.** Pvk2Pfx is a command-line tool that you install on the on-premises server. It requires access to the certificate exported as a file.

- **Windows PowerShell.** Version 5.0 or higher must be installed.
- **Az.Sql PowerShell module.**

Export a TDE certificate

You export the TDE certificate directly from the source SQL Server, or from the certificate store.

1. To locate the certificate details on the source SQL Server, in SQL Server Management Studio (SSMS), open a new query window, and connect it to the source SQL Server.
2. Run these Transact-SQL commands to list TDE protected databases and get the name of the certificate protecting encryption of the database to be migrated:

```
USE master
GO

SELECT db.name AS [database_name], cer.name AS [certificate_name]
FROM sys.dm_database_encryption_keys dek
LEFT JOIN sys.certificates cer ON dek.encryptor_thumbprint = cer.thumbprint
INNER JOIN sys.databases db ON dek.database_id = db.database_id
WHERE dek.encryption_state = 3
```

The output of the query gives you the name and certificate for each database.

3. To back up the TDE certificate, execute this script:

```
USE master
GO
BACKUP CERTIFICATE TDE_Cert
TO FILE = 'c:\full_path\TDE_Cert.cer'
WITH PRIVATE KEY (
    FILE = 'c:\full_path\TDE_Cert.pvk',
    ENCRYPTION BY PASSWORD = '<SomeStrongPassword>'
)
```

4. To copy the certificate to a Personal Information Exchange (.pfx) file, use PowerShell:

```
.\pvk2pfx -pvk c:/full_path/TDE_Cert.pvk
-pi "<SomeStrongPassword>"
-spc c:/full_path/TDE_Cert.cer
-pfx c:/full_path/TDE_Cert.pfx
```

5. Prepare the certificate for upload:

```
# Import the module into the Powershell session
Import-Module Az
# Connect to Azure with an interactive dialog for sign-in
Connect-AzAccount
# List subscriptions available and copy id of the
# subscription target managed instance belongs to
Get-AzSubscription
# Set subscription for the session (replace
# Guid_Subscription_Id with actual subscription id)
Select-AzSubscription Guid_Subscription_Id
```

6. Upload the 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-AzSqlManagedInstanceTransparentDataEncryptionCertificate -ResourceGroupName "<ResourceGroupName>"  

    -ManagedInstanceName "<ManagedInstanceName>"  

    -PrivateBlob $securePrivateBlob  

    -Password $securePassword
```

The certificate is now available to the specified managed instance, and the backup of the corresponding TDE protected database can be restored successfully.

Creating the 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. DMS is a free service that supports migrations of different databases to Azure database offerings. DMS can migrate MySQL, PostgreSQL, MariaDB databases to Azure Database for MySQL/PostgreSQL/MariaDB, and also supports SQL Server migrations, including SQL MI.

The service uses the Data Migration Assistant to generate assessment reports that provide recommendations to guide you through the changes required before migration. It's up to you to do any required remediation. When you're ready to begin migration, Azure DMS does all the required steps. You can fire and forget your migration projects, safe in the knowledge that the process follows Microsoft best practices.

Register the Microsoft.DataMigration resource provider

Before using the Data Migration Assistant, you must register a resource provider in Azure:

1. Sign in to the [Azure portal](#), select **All services**, and then select **Subscriptions**.
2. Select the subscription where you want to create the instance of the Azure Database Migration Service, and then select **Resource providers**.

The screenshot shows the Azure portal interface. On the left, a dark sidebar lists various services: Create a resource, All services, Favorites, Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, and Cost Management + The main area displays the 'Subscriptions' page under 'Microsoft'. It shows a search bar, role selection (Owner, 3 selected), and an 'Apply' button. Below is a table with columns 'SUBSCRIPTION...' and 'SUBSCRIPTION ID...'. A specific row is highlighted in blue, showing '<subscription>' and '<subscription ID>'. To the right, a detailed view of a subscription is shown with tabs for Overview, Access control (IAM), Diagnose and solve problems, COST MANAGEMENT + BILLING, Partner information, SETTINGS (with 'Resource providers' highlighted in red), Policies, Management certificates, My permissions, Properties, and Resource locks. The bottom section includes SUPPORT + TROUBLESHOOTING and a New support request button.

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

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

The screenshot shows the Azure portal's 'Resource providers' section for a specific subscription. On the left, there's a sidebar with various service icons. The main area has a search bar at the top. Below it, there are sections for 'Overview', 'Access control (IAM)', 'Diagnose and solve problems', 'COST MANAGEMENT + BILLING', 'Partner information', and 'SETTINGS'. Under 'SETTINGS', the 'Resource providers' section is expanded, showing a list with one item: 'Microsoft.DataMigration' with a status of 'NotRegistered' and a red-bordered 'Register' button.

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.

The screenshot shows the Azure portal's 'New' screen. The search bar at the top contains the text 'Azure Database Migration Service'. A dropdown menu below the search bar shows the result 'Azure Database Migration Service' highlighted with a red box. Other search results include 'Azure Database for MySQL', 'Azure Database for PostgreSQL', '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'. To the right of the search results, there are several preview icons and links for various services like Ubuntu Server, DevOps Project, Web App, SQL Database, Cosmos DB, Storage Account, and Serverless Function App.

2. On the Azure Database Migration Service screen, select **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 Azure region where you want to create the instance of DMS.
5. Select an existing virtual network or create a new one.
6. Select a SKU from the **Premium** pricing tier as this supports an online migration.

Migrate using the Azure Database Migration Service

To use the Azure Database Migration Service to migrate a database to Azure SQL Database managed instance, you must create a migration project and configure its details, such as the source database. Now you'll see how to migrate in detail.

Create a migration project

Now you have a migration service instance, you can add a migration project to it:

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, select the instance, and then select **+ New Migration Project**.
3. On the **New migration project** screen, specify a name for the project, and 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 **Online data migration**.
4. Select **Create and run activity** to create the project.

Specify source details

Next, specify the source of the data you want to migrate:

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. If a trusted certificate isn't installed, SQL Server generates a self-signed certificate when the instance is started. Use this certificate to encrypt the credentials for client connections.
3. Select **Save**.
4. On the **Select source databases** screen, select the database for migration.
5. Select **Save**.

Specify target details

Now, fix the managed instance where you want to send the data:

1. On the **Migration target details** screen, specify the **Application ID** and **Key** that the Migration Service instance uses to connect to Azure SQL Database managed instance and the Azure Storage account.
2. Select the **Subscription** that contains the target-managed instance.
3. Select the **Azure SQL Database managed instance**.
4. Provide **SQL User** and **Password** details to connect to the Azure SQL Database managed instance.
5. Select **Save**.

Select source databases

The next step is to configure the details of the database to migrate data from:

1. On the **Select source databases** screen, select the source database that you want to migrate.

Configure migration settings

Complete some migration settings before you migrate the data:

1. On the **Configure migration settings** screen, provide the following details:
 - **SMB network location share.** The local SMB network share that contains the full database backup files and transaction log backup files that the Azure Database Migration Service uses for migration.
 - **User name.** Make sure that the Windows user has full control privilege on the network share that you provided above.
 - **Password.** Password for the user.
 - **Subscription of the Azure Storage Account.** Select the subscription that contains the Azure Storage Account.
 - **Azure Storage Account.** Select the Azure Storage Account where DMS can upload the backup files from the SMB network share, and use for database migration.
2. Select **Save**.

Review the migration summary

The final configuration task is to review your setup:

1. On the **Migration summary** screen, in the **Activity name** text box, specify a name for the migration activity.
2. Review and verify the details associated with the migration project.

Run and monitor the migration

Now you're ready to execute the migration:

1. Select **Run migration**.
2. On the migration activity screen, select **Refresh** to update the display.

Performing migration cutover

After the full database backup is restored on the target instance of Azure SQL Database managed instance, the database is available for a migration cutover.

1. When you're ready to complete the online database migration, select **Start Cutover**.
2. Stop all the incoming traffic to source databases.
3. Take the tail-log backup, make the backup file available in the SMB network share, and then wait until this final transaction log backup is restored.
4. At that point, you'll see **Pending changes** set to 0.

5. Select **Confirm**, and then select **Apply**.
6. When the database migration status shows **Completed**, connect your applications to the new target instance of Azure SQL Database managed instance.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Migrate to SQL Database Managed Instance

Many migrations involve a period when the on-premises and the cloud database must be kept synchronized. For example, there might be a time when clients make changes to both databases.

You've migrated the sports retail products database into Azure SQL Database managed instance. The website is already using the cloud database. You're starting to reconfigure clients to use the new database. You've decided to move users to the new system in teams. For each team, you'll take time to resolve any problems before migrating the next users. Next, you'll reconfigure the data analysis system to use the new database in Azure. During this time, you want to ensure that the cloud and on-premises databases are synchronized every hour.

Here, you'll learn about methods you can use to implement synchronization.

Connectivity options with on-premises servers

Often, you want to keep data in on-premises databases synchronized with Azure SQL Database managed instance. You might want to stage the migration of client applications to the new database, for example, which means there's a period when clients connect to both databases.

Before you choose a data synchronization method, it's important to ensure you have connectivity that's secure. There are three different connectivity options available to establish communication between computers on-premises and resources in Azure.

- **Point-to-Site.** A Point-to-Site (P2S) VPN gateway connection lets you create a secure connection to your virtual network from an individual client computer.
- **Site-to-Site.** A Site-to-Site VPN gateway is used to connect an entire on-premises site to the Azure network.
- **ExpressRoute.** Azure ExpressRoute enables you to create private connections between Azure datacenters and on-premises infrastructure, or infrastructure in a colocation environment. ExpressRoute connections don't go over the public internet, and offer more reliability, faster speeds, lower latencies, and higher security than typical internet connections.

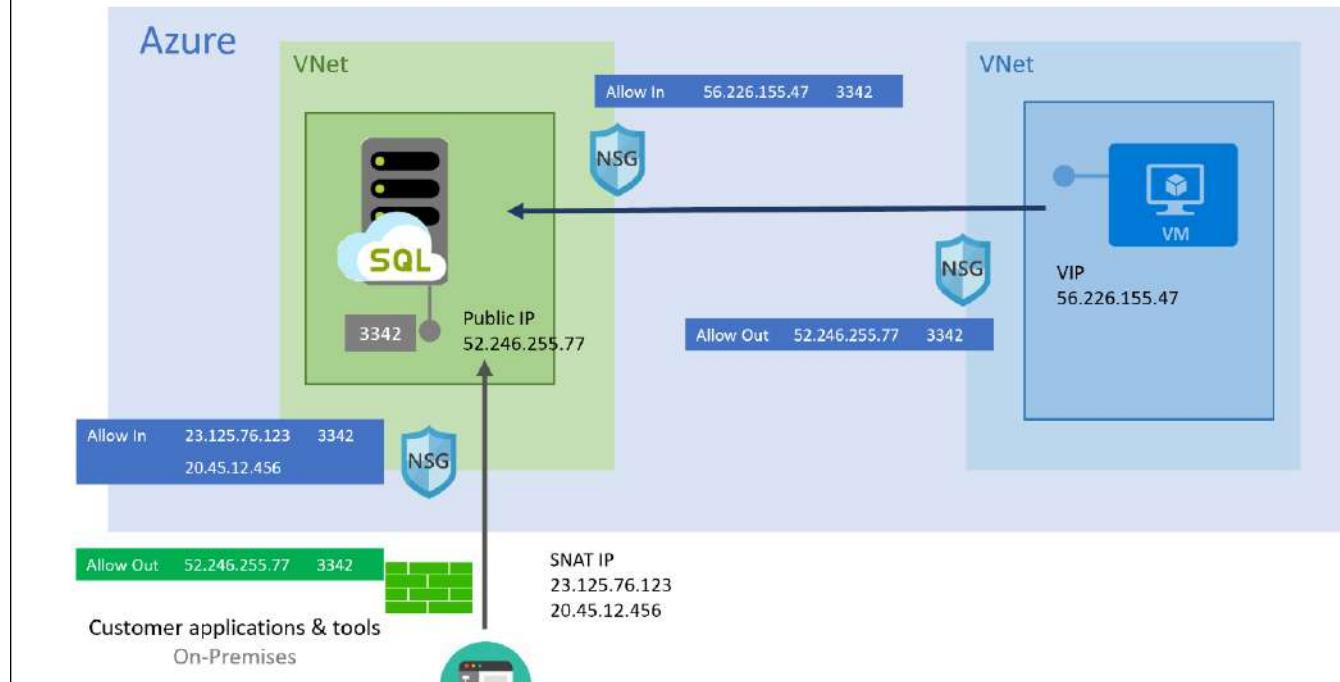
Azure SQL Database managed instance public endpoint secure access

Public endpoint for Azure SQL Database managed instance helps you connect to the database from the internet without using a VPN, and is designed for data communication only. Public endpoint for data can simultaneously coexist with the private endpoint. For security reasons, the implementation allows for Separation of Duties (SoD) between a database administrator and a network administrator when enabling the public endpoint.

To enable public endpoint for managed instance, two steps are required. For SoD, you'll need two separate roles, with the following database and network permissions, to complete these steps:

1. A database administrator who has RBAC permissions in scope Microsoft.Sql/managedInstances/* must run a PowerShell script to enable public endpoint for managed instance.
2. A network administrator who has RBAC permissions in scope Microsoft.Network/* must open the port 3342 used by the public endpoint on the network security group (NSG), and provide a UDR route to avoid asymmetric routing.

Public Endpoint - Secure Access



Choosing a synchronization method

You can use a number of methods to synchronize data from a SQL Database managed instance to an on-premises server and back.

NOTE: Although you could use backup and restore to move data from the cloud to on-premises databases, this isn't supported by Microsoft because managed instance is always at the latest version. You can't restore backups from the latest version of SQL Server to an earlier version.

BACPAC file using SqlPackage

A BACPAC file is simply a zipped version of your metadata and the data from your database. You can use this deployment method for Azure SQL Database, but managed instance doesn't support a migration using BACPAC in the Azure portal. Instead, you must use the SQLPackage utility, and the BACPAC file.

Bulk Copy Program (BCP)

BCP is a command-line tool that exports tables to files so you can import them. Use this approach to migrate from a single Azure SQL Database to Azure SQL managed instance and back.

SQL Server Integration Service (SSIS)

SSIS is primarily used for extract, transform, and load (ETL) tasks, but its control flow is robust enough to create a system level execution engine. The data flow is powerful enough to handle any volume of data transformation and manipulation tasks with auditing, debugging, and full source control support.

Azure Data Factory (ADF)

ADF is built for data movement and orchestration, with the focus on ingestion. ADF has the integration runtime support to run SSIS packages, and the public internet support for SQL Database managed instance.

Transactional replication

Transactional replication can copy data from your managed instance to any SQL Server. Transactional replication is a convenient approach for migrating data to and from a managed instance.

Import and export data with a BACPAC file

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 that contains the metadata and data from a SQL Server database. A BACPAC file can be stored in Azure Blob storage or in an on-premises location. The file can later be imported back into Azure SQL Database or a SQL Server on-premises installation. Use this method to restore SQL Server databases to Azure SQL Database and SQL Server IaaS virtual machines.

However, managed instance doesn't support migrating a database into an instance database from a BACPAC file using the Azure portal. You must instead use the **SQLPackage** utility, and the BACPAC file. SSMS and SQL Server Data Tools have the latest version of SQLPackage.

SQLPackage exports

The following SqlPackage command imports the AdventureWorks2008R2 database from local storage to an Azure SQL Database server called adworksserver20170403. This command creates a new database called AdWorksDatabase:

```
SqlPackage.exe /a:import  
/tcs:"Data Source=adworksserver20170403.database.windows.net;  
Initial Catalog=AdWorksDatabase;User Id=Admin;Password=<password>"  
/sf:Adventureworks2008R2.bacpac  
/p:DatabaseEdition=Premium  
/p:DatabaseServiceObjective=P6
```

To connect to a managed instance, you must have a point-to-site connection or an ExpressRoute connection.

SQLPackage imports

Using the following steps, you can import the BACPAC into an Azure SQL managed instance using SQLPackage:

1. Download and run the DacFramework.msi installer for Windows.
2. Open a new Command Prompt window, and run the following command.

```
cd C:\Program Files\Microsoft SQL Server\150\DAC\bin
```

3. Run the following command to import to the managed instance:

```
sqlpackage.exe /a:Import  
/TargetServerName:destinationdb.appname.database.windows.net  
/TargetDatabaseName:dbname /TargetUser:admin
```

```
/TargetPassword:<password>
/SourceFile:C:\Users\user\Desktop\backup150.bacpac
```

Synchronizing data using SSIS or Azure Data Factory

SSIS has long been the ETL tool of choice for migrating data from point A to point B. SSIS has powerful control flow and data flow capabilities with near limitless amounts of data manipulation, many data transformation options, the ability to execute in parallel, and other features. You can still run SSIS packages from SQL Server on-premises and from SQL Server virtual machines in the cloud.

Azure Data Factory is fully managed data-integration-as-a-service in the cloud. The managed compute infrastructure provides data connectors, data conversions, and column-mapping transformations. This infrastructure also has activity dispatching to run and monitor activities in other services like Azure Databricks and HDInsight.

Which technology should you use?

There are a couple of choices here. First, you can continue using your SSIS packages and have them run in the cloud by using Azure Data Factory. Secondly, you could create a new Azure Data Factory pipeline to execute your data movements.

SSIS catalog considerations

Consider hosting the SSIS catalog database in Azure SQL Database, with virtual network service endpoints, or in managed instance. This way, you can join your Azure-SSIS Integration Runtime (IR) to:

- The same virtual network.
- A different virtual network that has a network-to-network connection with the managed instance network.

If you host your SSIS catalog in Azure SQL Database with virtual network service endpoints, make sure you join your Azure-SSIS IR to the same virtual network and subnet. When you join your Azure-SSIS IR to the same virtual network as the managed instance, ensure that the Azure-SSIS IR is in a different subnet to the managed instance.

If you join your Azure-SSIS IR to a different virtual network than the managed instance, we recommend either virtual network peering, or a virtual network to virtual network connection. In all cases, the virtual network can only be deployed through the Azure Resource Manager deployment model.

Network security group

If you need to implement a network security group (NSG) for the subnet used by your Azure-SSIS integration runtime, allow inbound and outbound traffic through TCP ports (1433, 11000-11999, 14000-14999). This is because the nodes of your Azure-SSIS integration runtime in the virtual network use these ports to access SSIS DB hosted by your Azure SQL Database server. This requirement isn't applicable to SSISDB hosted by managed instance.

Synchronizing data with transactional replication

Transactional replication enables you to replicate data into an Azure SQL Database managed instance database from a SQL Server database. You can also use transactional replication to push changes made in a database in Azure SQL Database managed instance to a SQL Server database. Azure SQL Database managed instance is flexible because it can be a publisher, distributor, and subscriber.

In fact, one of the use cases for transactional replication with Managed Instance is the ability to migrate databases from one SQL Server or managed instance to another database by continuously publishing the changes. You have a publisher that has the source data. From there, you'll decide which tables and how much of the data to replicate.

Replication is one of the few technologies that allows you to replicate parts of a table. We refer to these table parts as "articles". This data is then sent to a distributor, which is a supplier of the data to any number of subscribers.

Azure SQL Database managed instance supports the following replication types:

- Transactional
- Snapshot
- One-way
- Bidirectional

Troubleshooting

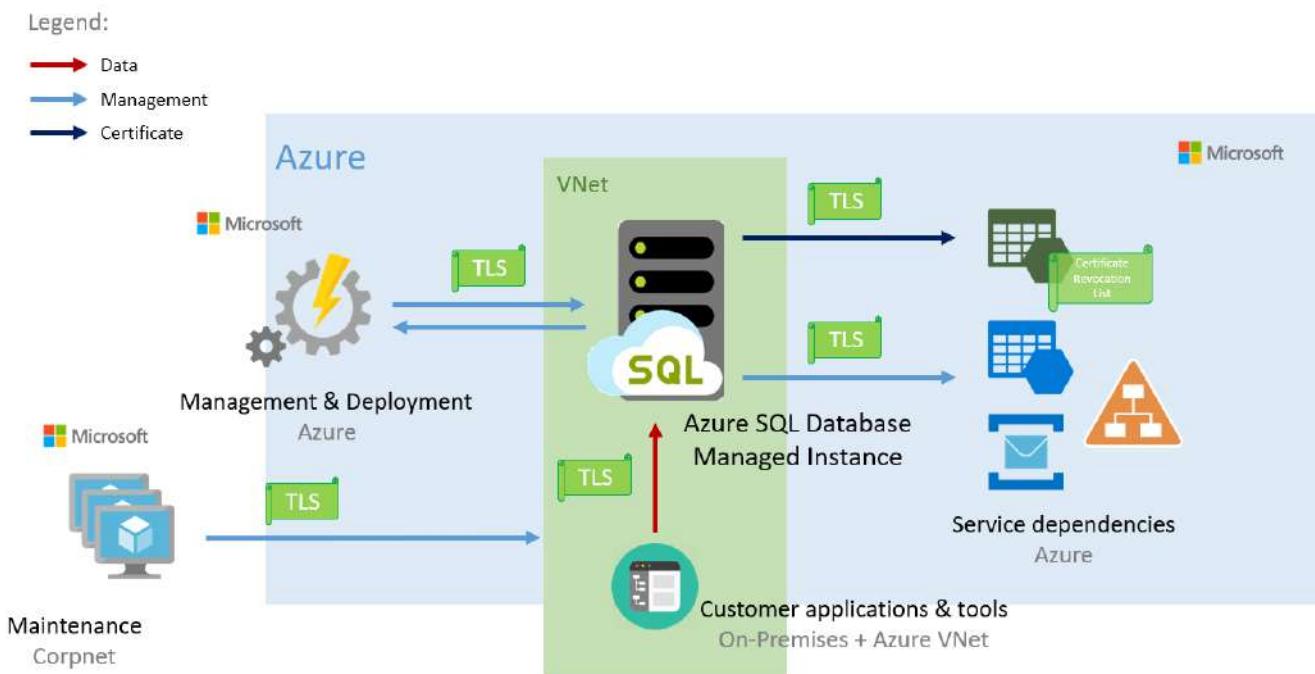
One of the most common issues customers contend with when setting up replication with Azure SQL Database managed instance is the network security layer. Managed instance is secure by default, so most ports and protocols can't access the managed instance virtual network. You need proper security access to ensure that replication components communicate with each other effectively. The distributor can be an Azure SQL Database managed instance, but it must be a version equal to, or higher than, the configured publisher.

Requirements

- Connectivity uses SQL Authentication between replication participants
- An Azure Storage Account share for the working directory used by replication
- Open port 445 (TCP outbound) in the security rules of the managed instance subnet to access the Azure file share
- Open port 1433 (TCP outbound) if the publisher or distributor is on a managed instance and the subscriber is on-premises

Connecting applications to a managed instance

An Azure SQL Database managed instance must be placed inside an Azure virtual network subnet that's dedicated to managed instances. This deployment gives you a secure private IP address and the ability to connect to on-premises networks.



Users and client applications can connect to the managed instance database through the Azure portal, PowerShell, Azure CLI, and the REST API.

Managed instances depend on Azure services such as Azure Storage for backups, Azure Event Hubs for telemetry, Azure Active Directory for authentication, and Azure Key Vault for Transparent Data Encryption (TDE).

The managed instances make connections to these services.

All communications are encrypted and signed using certificates. To check the trustworthiness of communicating parties, managed instances constantly verify these certificates through certificate revocation lists. If the certificates are revoked, the managed instance closes the connections to protect the data.

This document belongs to Srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Summary

In your sports clothing retail company, you needed an easy and quick way to migrate your products database into Azure. You chose Azure SQL Database Managed Instance because it provides a platform as a service (PaaS) implementation of SQL Server in the cloud. Azure SQL Database also has almost 100 percent compatibility with on-premises versions of SQL Server. You migrated the data, and ensured it was synchronized with the on-premises database so you could reconfigure client applications and the data analysis system.

Before Managed Instance was available, system architects who wanted to migrate a database to Azure had two options:

- You could use Azure SQL Database, which provided a full PaaS solution with the consequent reduction in administrative time. However, this often required significant modification of the database, because Azure SQL Database doesn't support all the features of SQL Server.
- You could install SQL Server on a virtual machine in Azure. However, this infrastructure as a service (IaaS) solution requires you to run, administer, and update the virtual machine, its operating system, and SQL Server yourself. You could expect to spend more time on administration.

Azure SQL Database Managed Instance provides a new solution that's a full PaaS but supports almost all the features of SQL Server. This solution gives you the low administrative load you'd expect from Azure SQL Database but without the need to rewrite your database and change your client apps.

Learn more

- [What is Azure SQL Database Managed Instance?](#)
- [SQL Server instance migration to Azure SQL Database Managed Instance](#)
- [Tutorial: Migrate SQL Server to an Azure SQL Database Managed Instance offline using DMS](#)
- [Differences between Compatibility Level 140 and Level 150](#)
- [Pvk2Pfx](#)

Introduction

Azure SQL Database has several authentication and authorization options which are different from the options in SQL Server. This is because Azure SQL Database and Azure SQL Managed Instance rely on Azure Active Directory instead of Windows Server Active Directory.

This module explores the practices of granting permissions and what the various permissions do within a database. This module also explores the concept of “least privilege”. While the built-in roles in SQL Server and other database engines provide broad brushes of security privileges, many applications need more granular security on database objects.

Learning objectives

In this module you will learn about:

- Describe the differences between Active Directory and Azure Active Directory
- Explain authentication Options for Azure SQL Database
- Describe security principals
- Explain object permissions
- Identify authentication and authorization failures

Describe Active Directory and Azure Active Directory

A common question is what is the difference between Azure Active Directory and Windows Server Active Directory, which we'll refer to simply as 'Active Directory'? This is especially confusing for new administrators because Azure Active Directory interacts with Active Directory. Both solutions provide authentication services and identity management, but in different ways—Active Directory uses a protocol called Kerberos to provide authentication using tickets, and it is queried by the Lightweight Directory Access Protocol (LDAP). Azure Active Directory uses HTTPS protocols like SAML and OpenID Connect for authentication and uses OAuth for authorization.

The two services have different use cases—for example, you cannot join a Windows Server to an Azure Active Directory domain and work together in most organizations to provide a single set of user identities. A service called Azure Active Directory Connect connects your Active Directory identities with your Azure Active Directory.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Describe authentication and identities

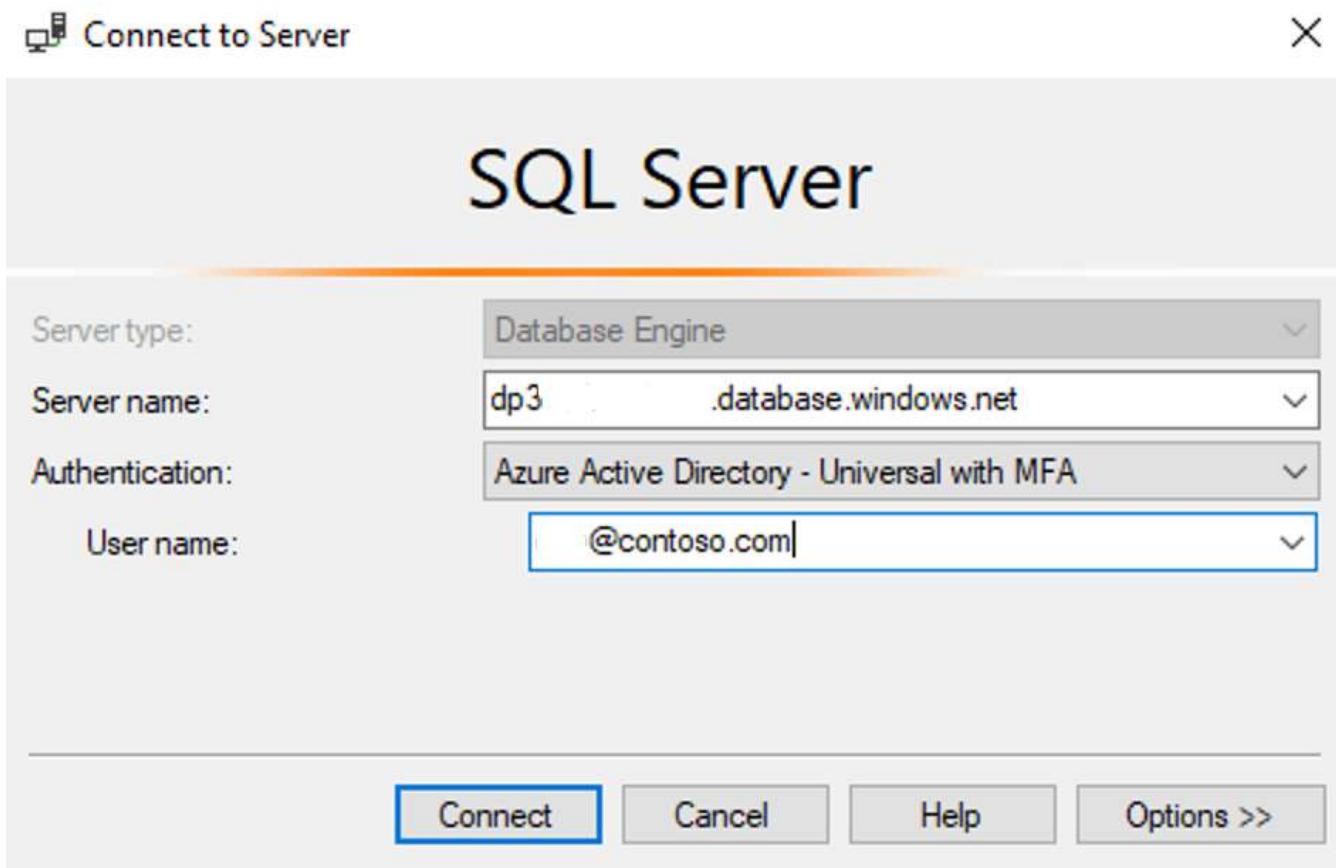
Both on-premises SQL Server installations and SQL Server installations within Azure Virtual Machines support two modes of authentication: SQL Server authentication and Windows Authentication. When using SQL Server authentication, SQL Server-specific login name and password information is stored within SQL Server, either in the master database, or in the case of contained users, within the user database. Using Windows Authentication, users connect to the SQL Server using the same Active Directory account they use to log into their computer (as well as accessing file shares and applications).

Active Directory authentication is considered to be more secure because SQL Server authentication allows for login information to be seen in plain text while being passed across the network. In addition, Active Directory authentication makes it easier to manage user turnover. If a user leaves the company and you use Windows authentication, the administrator would only have to lock the single Windows account of that user, instead of identifying each occurrence of a SQL login.

Azure SQL Database similarly supports two different modes of authentication, SQL Server authentication and Azure Active Directory authentication. SQL Server authentication is the same authentication method that has been supported in SQL Server since it was first introduced, where user credentials are stored within either the master database or the user database. Authentication via Azure Active Directory allows the user to enter the same username and password, which is used to access other resources such as the Azure portal or Microsoft 365.

As mentioned above, Azure Active Directory can be configured to sync with the on-premises Active Directory. This option allows users to have the same usernames and passwords to access on-premises resources as well as Azure resources. Azure Active directory adds on additional security measures by allowing the administrator to easily configure multi-factor authentication (MFA).

With MFA enabled on an account, after the correct username and password is supplied, a second level of authentication is required. By default, MFA can be configured to use the Windows Authenticator application, which will then send a push notification to the phone. Additional options for the default MFA action include sending the recipient a text message with an access code, or having the user enter an access code that was generated with the Microsoft Authenticator application. If a user has MFA enabled, they have to use the Universal Authentication with MFA option in Azure Data Studio and SQL Server Management Studio.



Both Azure SQL Database for SQL Server and Azure Database for PostgreSQL support configuring the server that is hosting the database to use Azure Active Directory Authentication.

The screenshot shows the Azure portal interface for an Azure SQL Server named 'dp300-lab06-xyz'. The 'Overview' tab is selected. Key details shown include:

- Resource group (change) : DP-300-RG
- Status : Available
- Location : East US
- Subscription (change) : Contoso Ltd
- Subscription ID :
- Tags (change) : Click here to add tags

On the right side, under 'Server admin', the account 'j' is listed. Under 'Active Directory admin', the account 'DBA Team' is listed and highlighted with a red box. The 'Server name' is listed as dp300-lab06-xyz.database.windows.net.

This login allows the admin access to all of the databases in the server. It is a best practice to make this account an Azure Active Directory group, so access is not dependent on a single login. In the above image, that group has been called DBA team. The Azure Active Directory Admin account grants special permissions and allows the account or group that holds that permission to have sysadmin like access to the server and all of the databases within the server. The admin account is only set using Azure Resource Manager and not at the database level. In order to change the account or group, you have to use the Azure portal, PowerShell, or CLI.

Describe Security Principals

Security principals are entities that can request SQL Server resources and to which you can (usually) grant permissions. There are several sets of security principals in SQL Server. Security principals exist at either the server level or the database level and can be either individuals or collections. Some sets have a membership controlled by the SQL Server administrators, and some have a fixed membership.

At the database level, we'll look at users, database roles, application roles.

NOTE: New logins can be added by administrators on Azure SQL Database, but new server roles cannot be created.

Schemas and SECURABLES

Before we look at the details of security principals, we need to understand the concepts of SECURABLES and schemas. SQL Server and Azure SQL Database have three scopes for SECURABLES. SECURABLES are the resources within the database to which the authorization system manages access. For example, a table is a SECURABLE. To simplify access control, SQL Server contains SECURABLES in nested hierarchies called SCOPES. The three SECURABLE SCOPES are the server, the database, and the schema. A schema is a collection of objects within your database, which allows objects to be grouped into separate namespaces.

Every user has a default schema. If a user tries to access an object without specifying a schema name, as in:

`SELECT name FROM customers`, it's assumed the object is in the user's default schema. If there's no such object in the default schema, SQL Server will check to see if the object is in the pre-defined dbo schema. If there's no object of the specified name in either the user's default schema, or in the dbo schema, the user will receive an error message. It's considered best practice to always specify the schema name when accessing objects, so the previous select would be something like: `SELECT name FROM SalesSchema.customers`. If a user hasn't been given a default schema, their default schema is set to dbo.

By default, if no schema is specified when a user creates an object, SQL Server will attempt to create it in the user's default schema. If the user hasn't been granted permission to create objects in their default schema, the object can't be created.

Logins and users

No matter the mode of authentication that is used, a login name used to access your SQL database is set up as a login within the instance. Those logins are set up at the instance level of SQL Server and stored in the master database. However, you can configure contained users, which are added at the database level. These users can be configured as SQL Server Authentication users as well as either Windows Authentication users or Azure Active Directory users (depending on which platform you're using). In order to create these users, the database must be configured for partial containment, which is configured by default in Azure SQL Database, and optionally configurable in SQL Server.

These users only have access to the database that the user is set up with. For the purposes of Azure SQL Database, it's considered a best practice to create users at the scope of the user database, and not in the master database as shown below.

```
CREATE USER [dba@contoso.com] FROM EXTERNAL PROVIDER;
GO
```

The `CREATE USER` statement is executed in the context of the user database. In the example above, the user is an Azure Active Directory user as indicated with the `FROM EXTERNAL PROVIDER` syntax.

If logins are created at the instance level in SQL Server, a user should then be created within the database, which maps the user to the server-based login as shown in the following example.

```
USE [master]
GO

CREATE LOGIN demo WITH PASSWORD = 'Pa55.w.rd'
GO

USE [wideworldimporters]
GO

CREATE USER demo FROM LOGIN demo
GO
```

The login is first created in the master database, and then in the WideWorldImporters database a user is created to map to that user. Logins are used to access the SQL Server or the Azure SQL Database, but to do any work within a database, the login must be mapped to a username. The username is used for all authentication.

Logins and usernames are the most important security principals you need to be aware of, but the next sections describe some of the other concepts and terms when dealing with authorization.

Database roles

As you can imagine, database security can get complicated for applications with many users. In order to make it easier for both administrators and auditors, most database applications use role-based security. Roles are effectively security groups that share a common set of permissions. Combining permissions into a role allows a set of roles to be created for a given application. Some examples of roles would be administrators who had full access to all of the databases and servers, reporting users who only read the database, and an application account that had access to write data into the database. The roles can be defined when the application is designed, and then users can be assigned to those roles as they need access to the database. Role-based access control is a common architecture across computer systems and is how authorization is managed in Azure Resource Manager.

SQL Server and Azure SQL Database both include built-in roles that are defined by Microsoft, and also provide the option to create custom roles. Custom roles can be created at the server or database level. However, server roles can't be granted access to objects within a database directly. Server roles are only available in SQL Server and Azure SQL Managed Instance, not in Azure SQL Database.

Within a database, permissions can be granted to the users that exist within the database. If multiple users all need the same permissions, you can create a database role within the database and grant the needed permissions to this role. Users can be added as members of the database role. The members of the database role will inherit the permissions of the database role.

```
CREATE USER [DP300User1] WITH PASSWORD = 'Pa55.w.rd'
GO

CREATE USER [DP300User2] WITH PASSWORD = 'Pa55.w.rd'
GO
```

```

CREATE ROLE [SalesReader]
GO

ALTER ROLE [SalesReader] ADD MEMBER [DP300User1]
GO

ALTER ROLE [SalesReader] ADD MEMBER [DP300User2]
GO

GRANT SELECT, EXECUTE ON SCHEMA::Sales TO [SalesReader]
GO

```

In the above example, you can see that two users are created, and then a role called SalesReader is created. The two new users are added to the newly created role, and then finally the role is granted **SELECT** and **EXECUTE** permissions on the *Sales* schema. Any user who is in that role can select from any object in the *Sales* schema, and execute any stored procedure in the schema.

Application roles

Application roles can also be created within a SQL Server database or Azure SQL Database. Unlike database roles, users aren't made members of an application role. An application role is activated by the user, by supplying the pre-configured password for the application role. Once the role is activated the permissions that are applied to the application role are applied to the user until that role is deactivated.

Built-in database roles

Microsoft SQL Server contains several fixed database roles within each database for which the permissions are predefined. Users can be added as members of one or more roles. These roles give their members a pre-defined set of permissions. These roles work the same within Azure SQL Database and SQL Server.

Database role	Definition
db_accessadmin	Allows users to create other users within the database. This role doesn't grant access to the schema of any of the tables, nor does it grant access to the data within the database.
db_backupoperator	Allows users to back up a database in a SQL Server or SQL Managed Instance. The role <i>db_backupoperator</i> doesn't confer any permissions in an Azure SQL Database.
db_datareader	Allows users to read from every table and view within the database.
db_datawriter	Allows users to INSERT , UPDATE , and DELETE data from every table and view within the database.
db_ddladmin	Allows users to create or modify objects within the database. Members of this role can change the definition of any object, of any type, but members of

Database role	Definition
	this role aren't granted access to read or write any data within the databases.
db_denydatareader	Users who need to be prevented from reading data from any object in the database, when those users have been granted rights through other roles or directly.
db_denydatawriter	Users who need to be prevented from writing data to any object in the database, when those users have been granted rights through other roles or directly.
db_securityadmin	Users who need to be able to grant access to other users within the database. Members of this role aren't granted access to the data within the database; however members of this role can grant themselves access to the tables within the database. Membership in this database role should be limited to only trusted users.
db_owner	Users who need administrative access to the database. Members of this role can perform any action within the database by default. However, unlike the actual database owner, who has the user name dbo , users in the db_owner role can be blocked from accessing data by placing them in other database roles, such as db_denydatareader , or by denying them access to objects. Membership in this database role should be limited to only trusted users.

All users within a database are automatically members of the **public** role. By default, this role has no permissions granted to it. Permissions can be granted to the public role, but you should consider carefully whether that is really something you want to do. Granting permissions to the public role would grant these permissions to any user, including the guest account, if the guest account was enabled.

The built-in database roles do meet the needs of many applications; however with applications that require more granular security (for example, when you only want to grant access to a specific subset of tables) a custom role is often a better choice.

NOTE: By default, users in roles like **db_owner** can always see all of the data in the database. Applications can take advantage of encryption options like **Always Encrypted** to protect sensitive data from privileged users.

Azure SQL Database has two roles that are defined in the master database of Azure SQL server.

Database Role	Definition
dbmanager	Allows its members to create extra databases within the Azure SQL Database environment. This role is the equivalent of the dbcreator fixed server role in an on-premises Microsoft SQL Server.

Database Role	Definition
loginmanager	Allows its members to create extra logins at the server level. This role is the equivalent of the securityadmin fixed server role in an on-premises Microsoft SQL Server.

Fixed server roles

In addition to database roles, SQL Server and Azure SQL Managed Instance both provide several fixed server roles. These roles assign permissions at the scope of the entire server. Server level principals, which include SQL Server logins, Windows accounts, and Windows group can be added into fixed server roles. The permissions for fixed server roles are predefined, and no new server roles can be added. The fixed server roles are:

Fixed server role	Definition
sysadmin	Allows its members to perform any activity on the server.
serveradmin	Allows its members to change server-wide configuration settings (for example Max Server Memory) and can shut down the server.
securityadmin	Allows its members to manage logins and their properties (for example, changing the password of a login). The members can also grant and revoke server and database level permissions. This role should be treated as being equivalent to the sysadmin role.
processadmin	Allows its members to kill processes running inside of SQL Server.
setupadmin	Allows its members to add and remove linked servers using T-SQL.
bulkadmin	Allows its members to run the BULK INSERT T-SQL statement.
diskadmin	Allows its members to have the ability to manage backup devices in SQL Server.
dbcreator	Allows its members to create, restore, alter, and drop any database.
public	Every SQL Server login belongs to the public user role. Unlike the other fixed server roles, permissions can be granted, denied, or revoked from the public role.

Describe database and object permissions

All Relational Database Management platforms have four basic permissions, which control data manipulation language (DML) operations. These permissions are **SELECT**, **INSERT**, **UPDATE**, and **DELETE**, and they apply to all SQL Server platforms. All of these permissions can be granted, revoked or denied on tables and views. If a permission is granted using the **GRANT** statement, then the permission is given to the user or role referenced in the **GRANT** statement. Users can also be denied permissions using the **DENY** command. If a user is granted a permission and denied the same permission, the **DENY** will always supersede the grant, and the user will be denied access to the specific object.

```
GRANT SELECT ON dbo.Company TO Demo
GO
DENY SELECT ON dbo.Company TO Demo
GO
EXECUTE AS USER = 'Demo'
SELECT Name, Address FROM dbo.Company
```

```
Msg 229, Level 14, State 5, Line 17
The SELECT permission was denied on the object 'Company', database 'WideWorldImporters', schema 'dbo'.
```

Completion time: 2020-05-13T14:42:28.8361616-07:00

In the above example, the user **Demo** is granted **SELECT** and then denied **SELECT** permissions on the **dbo.Company** table. When the user tries to execute a query that selects from the **dbo.Company** table, the user receives an error that **SELECT** permission was denied.

Table and view permissions

Tables and views represent the objects on which permissions can be granted within a database. Within those tables and views, you can additionally restrict the columns that are accessible to a given security principal (user or login). SQL Server and Azure SQL Database also include row-level security, which can be used to further restrict access.

Permission	Definition
SELECT	Allows the user to view the data within the object (table or view). When denied, the user will be prevented from viewing the data within the object.
INSERT	Allows the user to insert data into the object. When denied, the user will be prevented from inserting data into the object.
UPDATE	Allows the user the update data within the object. When denied, the user will be prevented from updating data in the object.

Permission	Definition
DELETE	Allows the user to delete data within the object. When denied, the user will be prevented from deleting data from the object.

Azure SQL Database and Microsoft SQL Server have other permissions, which can be granted, revoked or denied as needed.

Permission	Definition
CONTROL	Grants all rights to the objects. It allows the user who has this permission to perform any action they wish against the object, including deleting the object.
REFERENCES	Grants the user the ability to view the foreign keys on the object.
TAKE OWNERSHIP	Allows the user the ability to take ownership of the object.
VIEW CHANGE TRACKING	Allows the user to view the change tracking setting for the object.
VIEW DEFINITION	Allows the user to view the definition of the object.

Function and stored procedure permissions

Like tables and views, functions and stored procedures have several permissions, which can be granted or denied.

Permission	Definition
ALTER	Grants the user the ability to change the definition of the object.
CONTROL	Grants the user all rights to the object.
EXECUTE	Grants the user the ability to execute the object.
VIEW CHANGE TRACKING	Allows the user to view the change tracking setting for the object.
VIEW DEFINITION	Allows the user to view the definition of the object.

EXECUTE AS

The `EXECUTE AS [user name]`, or `EXECUTE AS [login name]` (only available in SQL Server and Azure SQL Managed Instance) commands allow for the user context to be changed. As subsequent commands and statements will be executed using the new context with the permissions granted to that context.

If a user has a permission and the user no longer needs to have that permission, permissions can be removed (either grants or denies) using the REVOKE command. The revoke command will remove any `GRANT` or `DENY` permissions for the right specified to the user specified.

Ownership Chains

A concept called chaining applies to permissions, which allows users to inherit permissions from other objects. The most common example of chaining is a function or stored procedure that accesses a table during its execution. If the procedure has the same owner as the table, the stored procedure is able to be executed and access the table, even though the user doesn't have rights to access the table directly. This access is available because the user inherits the rights to access the table from the stored procedure, but only during the execution of the stored procedure, and only within the context of the stored procedures execution.

In the example below, run as a database owner or server administrator, a new user is created and added as a member of a new `SalesReader` role, which is then granted permission to select from any object and execute any procedure in the `Sales` schema. A stored procedure is then created in the `Sales` schema that accesses a table in the `Production` schema.

The example then changes content to be the new user and an attempt is made to select directly from the table in the `Production` schema.

```
USE Adventureworks2016;
GO

CREATE USER [DP300User1] WITH PASSWORD = 'Pa55.w.rd';
GO

CREATE ROLE [SalesReader];
GO

ALTER ROLE [SalesReader] ADD MEMBER [DP300User1];
GO

GRANT SELECT, EXECUTE ON SCHEMA::Sales TO [SalesReader];
GO

CREATE OR ALTER PROCEDURE Sales.DemoProc
AS
SELECT P.Name,
       SUM(SOD.LineTotal) AS Totalsales,
       SOH.OrderDate
  FROM Production.Product P
     INNER JOIN Sales.SalesOrderDetail SOD ON (SOD.ProductID = P.ProductID)
     INNER JOIN Sales.SalesOrderHeader SOH ON (SOH.SalesOrderID = SOD.SalesOrderID)
 GROUP BY P.Name,
          SOH.OrderDate
 ORDER BY Totalsales DESC;
```

```
GO
```

```
EXECUTE AS USER = 'DP300User1';

SELECT P.Name,
       SUM(SOD.LineTotal) AS Totalsales,
       SOH.OrderDate
  FROM Production.Product P
    INNER JOIN Sales.SalesOrderDetail SOD ON (SOD.ProductID = P.ProductID)
    INNER JOIN Sales.SalesorderHeader SOH ON (SOH.SalesorderID = SOD.SalesorderID)
 GROUP BY P.Name,
          SOH.OrderDate
 ORDER BY Totalsales DESC;
```

The above query results in an error that the user *DP300User1* doesn't have `SELECT` permission, because the role that the user belongs to doesn't have any privileges in the *Production* schema. Now we can try to execute the stored procedure:

```
EXECUTE AS USER = 'DP300User1';

EXECUTE Sales.DemoProc;
```

The *DP300User1* user has `EXECUTE` permission on the stored procedure in the *Sales* schema, because the user's role has `EXECUTE` permission on the *Sales* schema. Because the table has the same owner as the procedure, we have an unbroken ownership chain, and the execution will succeed and results will be returned.

Permission changes don't apply when dynamic SQL is being used within stored procedures. The reason that dynamic SQL breaks the permission chain is because the dynamic SQL is executed outside of the context of the calling stored procedure. You can see this behavior by changing the above stored procedure to execute using dynamic SQL as shown below.

```
CREATE OR ALTER PROCEDURE Sales.DemoProc
AS
DECLARE @sqlstring NVARCHAR(MAX)

SET @sqlstring = '
SELECT P.Name,
       SUM(SOD.LineTotal) AS Totalsales,
       SOH.OrderDate
  FROM Production.Product P
    INNER JOIN Sales.SalesOrderDetail SOD ON (SOD.ProductID = P.ProductID)
    INNER JOIN Sales.SalesOrderHeader SOH ON (SOH.SalesOrderID = SOD.SalesOrderID)
 GROUP BY P.Name, SOH.OrderDate'

EXECUTE sp_executesql @sqlstring
GO

-- 

EXECUTE AS USER = 'DP300User1'

EXECUTE Sales.DemoProc
```

The `DP300User1` user will receive an error that the user doesn't have `SELECT` permission on the `Production.Product` table, just like the user tried to execute the query directly. Permission chains don't apply and the user account that is executing the dynamic SQL must have rights to the tables and views that are being used by the code within the dynamic SQL.

Principle of least privilege

The principle of least privilege is fairly simple. The basic idea behind the concept is that users and applications should only be given the permissions needed in order for them to complete the task. Applications should only have permissions that they need to do in order to complete the task at hand.

As an example, if an application accesses all data through stored procedures, then the application should only have the permission to execute the stored procedures, with no access to the tables.

Dynamic SQL

Dynamic SQL is a concept where a query is built programmatically. Dynamic SQL allows T-SQL statements to be generated within a stored procedure or a query itself. A simple example is shown below.

```
SELECT 'BACKUP DATABASE ' + name + ' TO DISK = ''\\backup\\sql1\\' + name + '.bak'''  
FROM sys.databases
```

The above statement will generate a list of T-SQL statements to back up all of the database on the server. Typically, this generated T-SQL will be executed using `sp_executesql` or passed to another program to execute.

Identify authentication and authorization failures

A connection failure can result from reconfiguration, firewall settings, connection timeouts, or incorrect login information. Furthermore, if some Azure SQL Database or SQL Managed Instance resources are over capacity, you will not be able to connect.

Transient fault

When heavy workloads increase in the SQL Database service, the Azure infrastructure is able to dynamically reconfigure servers, and the client application may lose connection to the database during this operation.

Transient faults occur during database reconfiguration of a planned event or an unplanned event. These events are generally brief and shouldn't take longer than 60 seconds to complete.

Below is a list of a few transient errors that applications may receive when connecting to Azure SQL Database:

- Cannot open database “%.*ls” requested by the login. The login failed.
- Cannot process request. Not enough resources to process request.
- Cannot process request. Too many operations in progress for subscription “%ld”.

NOTE: For a complete list of transient errors, see [Troubleshooting connectivity issues and other errors with Azure SQL Database and Azure SQL Managed Instance](#).

How to monitor transient connectivity errors

Error	Action
Login failures	Look for any outages during the time when the application reported the errors at Microsoft Azure Service Dashboard.
Database reaches resource limits	Monitor your database's compute and storage resources carefully, and take action when it reaches its resource limits to prevent transient failures.
Extended authentication failures	File an Azure support request through the Azure portal if your application encounters connectivity error for longer than 60 seconds or if it occurs more than once in a given day.

Retry logic

Application developers should anticipate periodic transient failures when integrating with cloud services, like Azure SQL Database, and implement a retry logic instead of displaying application errors to users. Setting a maximum number of retries before the program terminates is also important.

We recommend waiting for 5 seconds at a minimum on your first retry. Each sequential retry should increase the delay exponentially, up to a maximum of 60 seconds.

NOTE: If a SELECT statement fails with a transient error for SQL Database or SQL Database Managed Instance, don't directly retry it. Instead, retry the SELECT statement in a new connection.

Unable to log in to the server

When the error **Login failed for user '< User name >'** happens, the service administrator can follow the following steps:

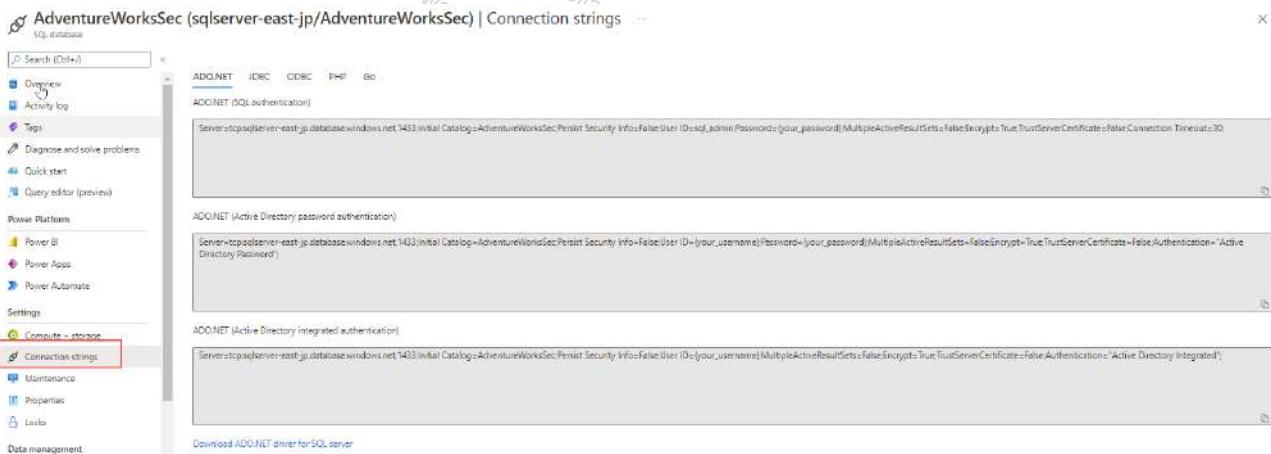
1. Check if the login is disabled by using the `sys.sql_logins` catalog view.
2. If the login is disabled, run `ALTER LOGIN <User name> ENABLE;` to enable it.
3. If the login does not exist, create it using the `CREATE LOGIN` statement.
4. Connect to the database you want to grant the user access to, and run the `CREATE USER` statement.
5. Either assign the user a role using the `ALTER ROLE` command, or grant the user access to one or more database objects using the `GRANT` command.

Connection string

When you receive connectivity errors, it is a good practice to make sure your connection string is working properly. This is mostly important when provisioning a new database, or after making infrastructure changes to a database service.

The Azure portal allows you to retrieve the connection string you need to interact with Azure SQL Database.

1. From the Azure portal, select **All services**, and then **SQL databases**. Filter and select your database.
2. On the blade for your database, select **Connection strings**.



3. Copy and edit the connection string by including your password, or replacing the server name as needed.
4. Reference the connection string updated in the client application.

To learn more about connectivity errors for Azure SQL Database and Azure SQL Managed Instance, see [Troubleshooting connectivity issues and other errors with Azure SQL Database and Azure SQL Managed Instance](#).

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which protocol is used by Azure Active Directory for Authorization?

- Kerberos
- LDAP
- OAuth

Check Answers

Multiple choice

Which database stores the information about logins in SQL Server?

- master
- model
- msdb

Check Answers

Multiple choice

Which role allows users to create users within a database?

- db_datareader
- db_accessadmin
- db_securityadmin

Check Answers

Multiple choice

Which permission allows the user to perform any option against a database object?

- Control
- Delete
- View Definition

[Check Answers](#)

Multiple choice

What feature allows a user to execute a stored procedure without having permission to access the tables referenced in the stored procedure?

- Ownership chaining
- Principle of least privilege
- Granular security

[Check Answers](#)

Summary

Azure SQL Database has several authentication and authorization options which are different from the options in SQL Server. This is because Azure SQL Database and Azure SQL Managed Instance rely on Azure Active Directory instead of Windows Server Active Directory.

This module explored the practices of granting permissions and what the various permissions do within a database. This module also explored the concept of “least privilege”. While the built-in roles in SQL Server and other database engines provide broad brushes of security privileges, many applications need more granular security on database objects.

Now that you've reviewed this module, you should be able to:

- Describe the differences between Active Directory and Azure Active Directory
- Explain authentication Options for Azure SQL Database
- Describe security principals
- Explain object permissions
- Identify authentication and authorization failures

Introduction

This module explores the encryption options available within Microsoft SQL Server, Azure SQL Database and Azure SQL Managed Instance. Each of the various platforms support different database encryption options. In this module students will explore these data encryption options and how to configure them.

Consider the following three scenarios when evaluating data encryption methods:

Scenario	Definition
Data at rest	Encrypting it while it's on file storage.
Data in transit	Encrypting it while it travels through private or public network communication channels.
Data in use	Encrypting it while it's in RAM or CPU caches.

Learning objectives

After taking this module, you will:

- Understand the difference between server and database firewall rules in Azure SQL Database
- Understand the data encryption options available in the various platforms
- Understand the role of Azure Key Vault in Transparent Data Encryption
- Explain how to enable encrypted connections

Explore Transparent Data Encryption

Microsoft SQL Server's Transparent Data Encryption (TDE) encrypts all the data within a target database at the page level. The data is encrypted as the data is written to the data page on disk and decrypted when the data page is read into memory. The end result is that all data pages on disk are encrypted.

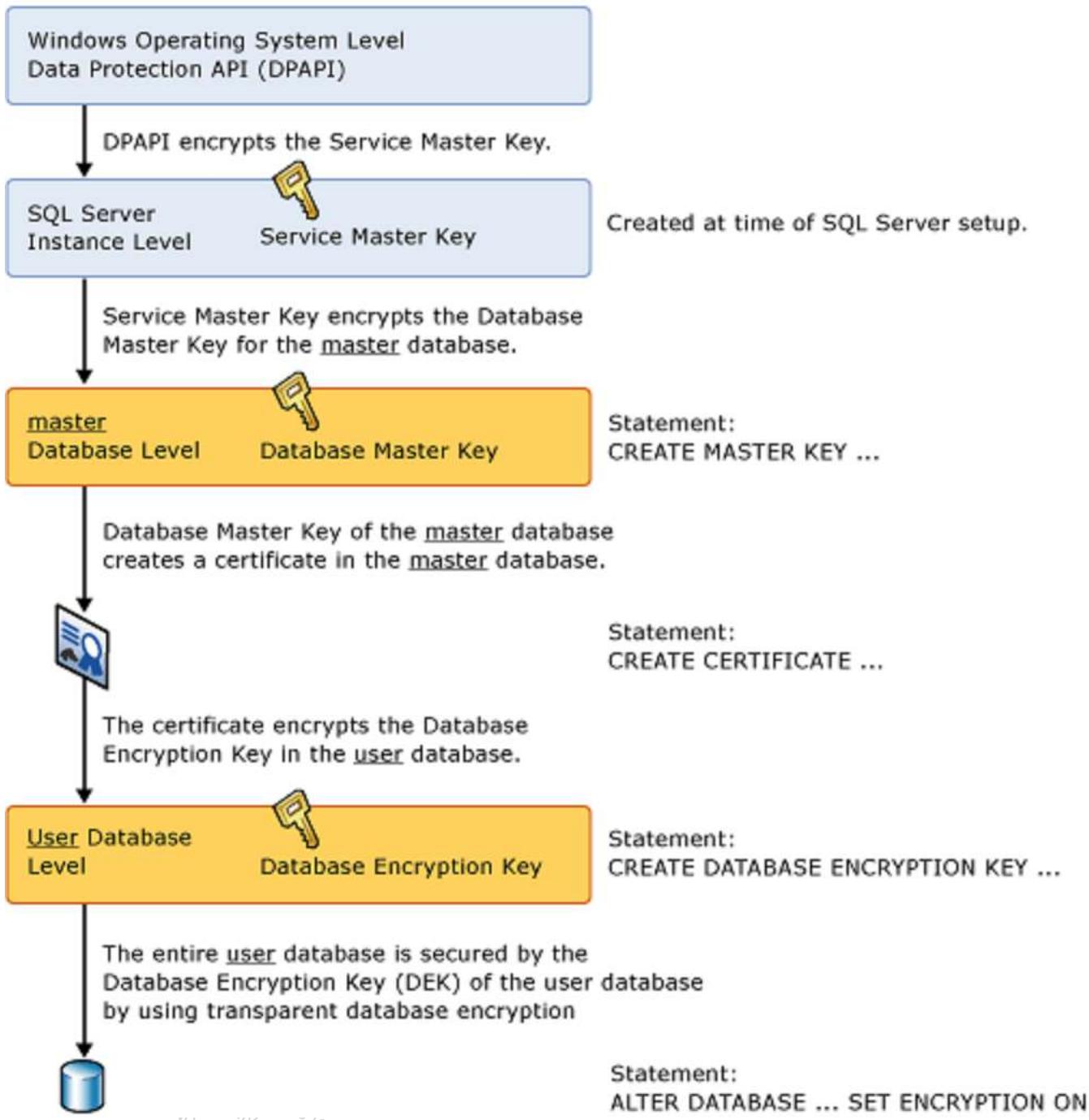
TDE doesn't encrypt data at the table or column level. Anyone with the appropriate permissions can read the data, copy the data and even share the data. Encryption at rest provides protection against someone restoring a backup to an unsecured server or making a copy of all the database and transaction log files and attaching them to another unsecured server. No decryption is done during the backup operation.

TDE protects data at rest, and follows several laws, regulations, and guidelines established in various industries. With this capability, software developers can encrypt data using the AES and 3DES encryption algorithms without having to change existing applications.

The following image shows how TDE encryption works.



Transparent Database Encryption Architecture



With Azure SQL Database, enabling TDE is simple. Databases that are created in Azure SQL Database after May 2017 have TDE enabled automatically. Databases that were created before May 2017 will have TDE disabled by default and TDE will need to be manually enabled on these databases. TDE is enabled in databases created after February 2019 with Azure SQL Managed Instance. Databases created before February 2019 will have TDE disabled.

Enabling TDE within an Azure SQL Database database is simply a matter of editing the database within the Azure portal. From the **Transparent data encryption** pane, select to enable data encryption.

 Save  Discard  Feedback

Transparent data encryption encrypts your databases, backups, and logs at rest without any changes to your application. To enable encryption, go to each database.

[Learn more](#)

Data encryption

 ON  OFF

Encryption status

 Encrypted

By default, databases within Azure SQL Database are encrypted using a Microsoft provided certificate. Microsoft Azure does provide a Bring Your Own Key option, which allows you to use a certificate that was created by your company and uploaded to Azure. If your company removes the certificate from Azure, then the database connections will be closed, and there will be no access to the database.

Enabling TDE within a Microsoft SQL Server Database is an easy process as only a few T-SQL commands are required. This process involves the following steps:

1. Set a master key within the master database using the `CREATE MASTER KEY ENCRYPTION` command.
2. Create a certificate in the master database, which will be used for the encryption using the `CREATE CERTIFICATE` command.
3. Create a database encryption key within the database, which allows you to enable TDE with the `CREATE DATABASE ENCRYPTION KEY` command.
4. Once the encryption key is created, it needs to be enabled using the `ALTER DATABASE` command.

The entire set of commands is shown below.

```
USE master;
GO

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa55.w.rd';
GO

CREATE CERTIFICATE MyServerCert
    WITH SUBJECT = 'TDEDemo_Certificate';
GO

USE [TDE_Demo];
GO

CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_256 ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
GO
```

```
ALTER DATABASE TDE_Demo SET ENCRYPTION ON;  
GO
```

Once TDE is enabled, it will take some time in order to encrypt the database as each database page must be read, encrypted and written back to disk. The larger the database the longer this operation will take. This process is a background process and is run at a low priority in order to not overload the IO or the CPU of the system.

Once the certificate that will be used by TDE has been created, it must be manually backed up and stored in a safe place. SQL Server integrates with Enterprise Key Managers (EKMs) in order to manage encryption keys. An example of an EKM is Azure Key Vault.

Managing the certificate is important, because if the certificate is lost and the database needs to be restored from a backup, the restore will fail, as the database can't be read.

NOTE: To use TDE with databases in an Always On Availability Group, the certificate used to encrypt the database must be backed up and restored to the other servers within the Availability Group that will be hosting copies of the database.

Azure disk encryption

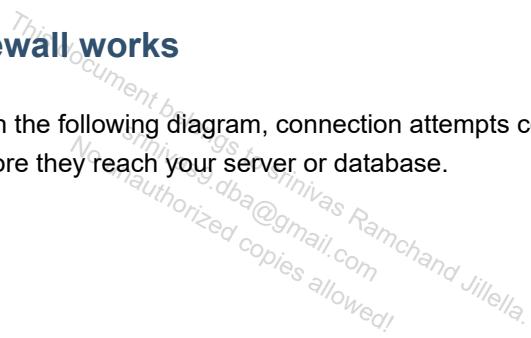
In addition to these SQL Server security features, Azure VMs include an extra layer of security, Azure Disk Encryption—a feature that helps protect and safeguard data and meet organization and compliance commitments. If you're using TDE, your data is protected by multiple layers of encryption with Azure Disk Encryption.

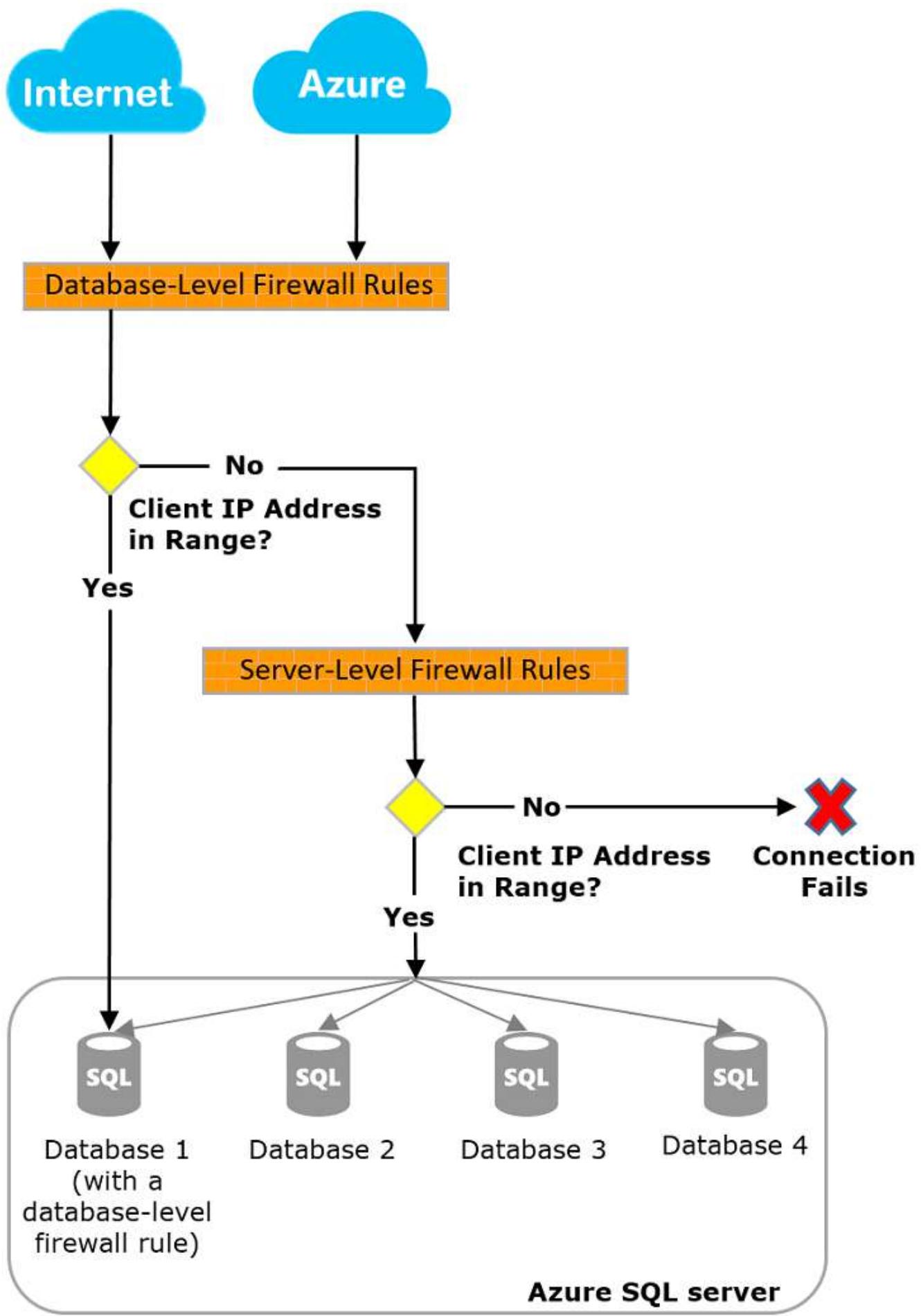
Configure server and database firewall rules

Firewalls are used to prevent unauthorized users from accessing protected resources. Each Azure SQL Database maps to a public IP address, which is hosted by Microsoft. Each Azure region will have one or more public IP addresses where you can reach your database gateway, which will then take you to your database.

How firewall works

As shown in the following diagram, connection attempts coming from the internet and Azure must go through the firewall before they reach your server or database.





As we can see in the image above, Azure provides built-in firewalls to limit access in order to protect your database and your data. In Azure SQL Database there are two distinct sets of firewall rules: server-level firewall rules and database-level firewall rules.

Server-level firewall rules

Both server and database level firewalls use IP Address rules instead of SQL Server Logins, and allow all users at the same public IP Address to access the SQL Server. For most companies, this is their outbound IP address.

Server-level firewalls are configured to allow users to connect to all databases on the server. Database level firewalls are used to grant or block specific IP Addresses from accessing specific databases.

Server level firewall rules can be configured using the Azure portal or using the `sp_set_firewall_rule` stored procedure from within the *master* database.

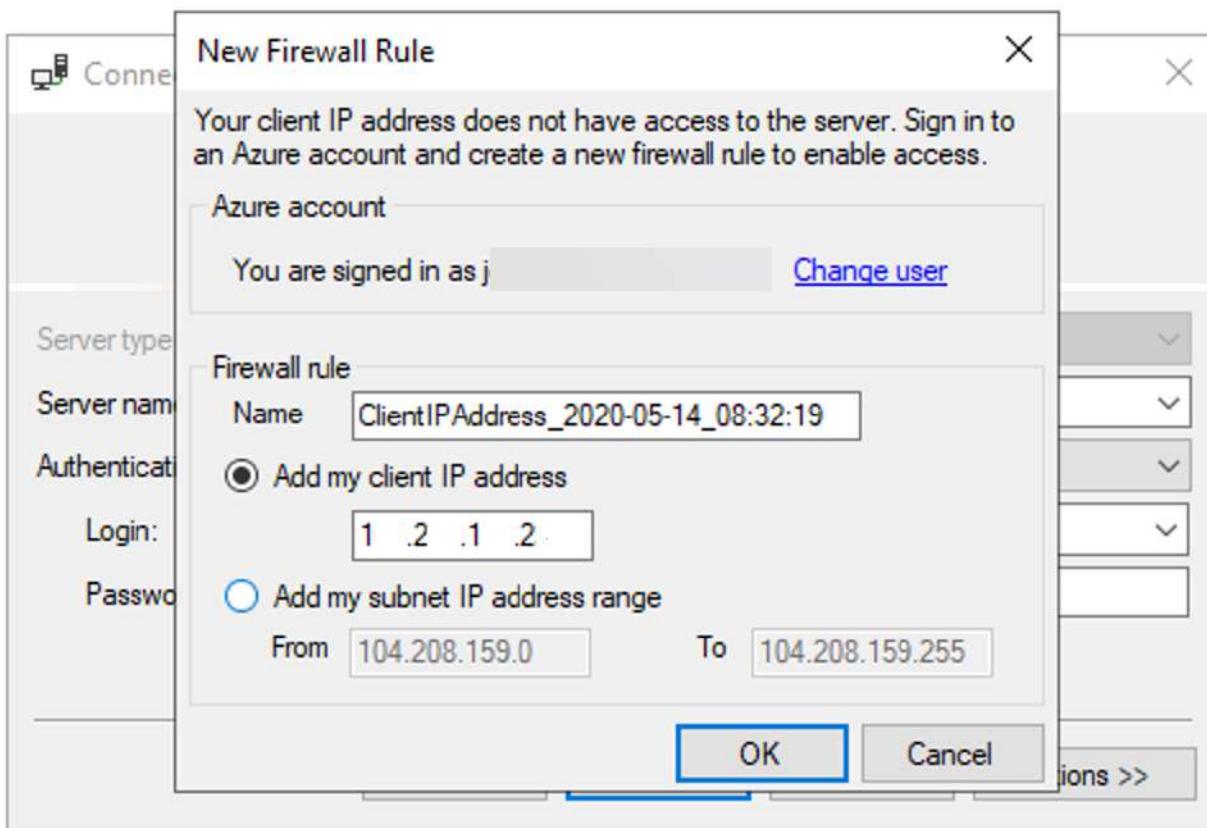
NOTE: The **Allow Azure Services and resources to access this server** server setting counts as a single firewall rule when enabled.

Database-level firewall rules

Database-level firewall rules are configured through T-SQL only using the `sp_set_database_firewall_rule` stored procedure from within the user database.

Upon connection, Azure SQL Database will look first for a database-level firewall rule for the database name specified in the connection string. If it does not exist, the firewall will then check the server-level IP firewall rules. Server-level IP firewall rules apply to all databases on the server. If either of these exist, the connection will be completed.

If neither exist and the user is connecting through SQL Server Management Studio or Azure Data Studio, they'll be prompted to create a firewall rule as shown below.



Virtual network endpoints

Virtual network endpoints allow traffic from a specific Azure Virtual Network. These rules apply at the server level, not just the database level.

Additionally, the service endpoint applies to only one region, which is the underlying endpoint's region.

An extra concern is that the virtual network that is connecting to the Azure SQL Database must have outbound access to the public IP address for Azure SQL Database, which can be configured using service tags for Azure SQL Database.

Private link

The Private Link feature allows you to connect to Azure SQL Database and other PaaS offerings using a private endpoint.

A private endpoint allows for a connection to your Azure SQL Database to go entirely over the Azure backbone network and not over the public internet.

This feature provides a private IP address on your Virtual Network. Another feature of private link is that it allows for Azure Express Route connections through that circuit.

Private link offers several benefits including cross-region private connectivity and protection against data leakage by only allowing connections to specific resources.

Explain object encryption and secure enclaves

In addition to supporting encryption at rest, SQL Server supports encrypting data within columns using Always Encrypted. Once data is encrypted, the application accessing the database must have the correct certificate in order to view the plain text values of the data.

Always Encrypted

Always Encrypted allows for the encryption of data within the client application, protecting sensitive data from malware and high-privileged users, such as Database Administrators (DBAs), server admins, cloud admins, or those who manage the data but should have no access. This encryption happens automatically based on the settings within the Microsoft SQL Server database, which tell the application what the encryption settings on the database column are.

The following table provides some scenarios for Always Encrypted usage:

Scenario	Definition
Client and data on-premises	For scenarios where you need to protect your on-premises database from high-privileged users, for example, external vendors managing SQL Server.
Client on-premises with data in Azure	In this scenario, to ensure Microsoft cloud administrators have no access to the data, Always Encrypted keys are stored in key store hosted on-premises, for SQL Database or SQL Server running in a virtual machine on Microsoft Azure.
Client and Data in Azure	In this scenario, the environment is fully hosted on Azure. While Always Encrypted doesn't completely isolate data from cloud administrators, the customer still benefits from the fact that the data is encrypted in the database.

Always Encrypted is based on a master encryption key and a column encryption key. Having both keys allows each column to be encrypted using a different encryption key for maximum data protection. Always Encrypted has various key stores that can store the certificate used by encryption.

Here's an example of enabling Always Encrypted. As shown below, you can see that *NationalIDNumber* and *BirthDate* columns are both in plain text.

```

SELECT
    [NationalIDNumber]
    ,[LoginID]
    ,[OrganizationNode]
    ,[OrganizationLevel]
    ,[JobTitle]
    ,[BirthDate]
FROM [AdventureWorks2017].[HumanResources].[Employee]

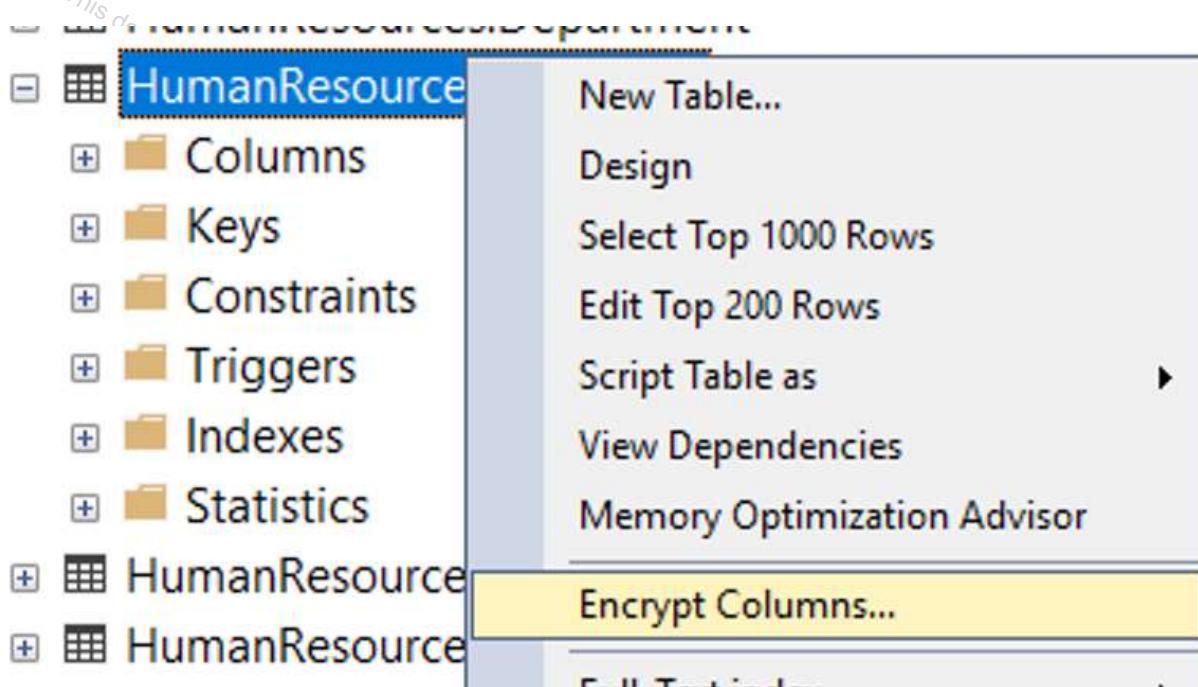
```

100 %

Results Messages

	NationalIDNumber	LoginID	OrganizationNode	OrganizationLevel	JobTitle	BirthDate
1	295847284	adventure-works\ken0	NULL	NULL	Chief Executive Officer	1969-01-29
2	245797967	adventure-works\terri0	0x58	1	Vice President of Engineering	1971-08-01
3	509647174	adventure-works\roberto0	0x5AC0	2	Engineering Manager	1974-11-12
4	112457891	adventure-works\rob0	0x5AD6	3	Senior Tool Designer	1974-12-23
5	695256908	adventure-works\gail0	0x5ADA	3	Design Engineer	1952-09-27
6	998320692	adventure-works\jossef0	0x5ADE	3	Design Engineer	1959-03-11
7	134969118	adventure-works\dylan0	0x5AE1	3	Research and Development Manager	1987-02-24
8	811994146	adventure-works\diane1	0x5AE158	4	Research and Development Engineer	1986-06-05

The next few images will show you how we can encrypt both of these columns using Always Encrypted. The encryption could be done using T-SQL, but in this example, you'll see the wizard from SQL Server Management Studio. You can reach the wizard by right-clicking on the table name in Object Explorer as shown below.



When you select **Encrypt Columns...**, the wizard will launch.

In the image below, you'll see the Always Encrypted launch screen. Select **Next** to choose the columns you want to encrypt.

In the image above, there are two different types of encryption specified. The *NationalIDNumber* column is encrypted with **Deterministic** encryption, and the *BirthDate* column is encrypted using **Randomized** encryption.

Randomized encryption is more secure than Deterministic encryption, but is more limited. The type of encryption can't be changed after the column is created. It's recommended to use Randomized encryption for columns that had a few well-known distinct values that could potentially be guessed by someone with access to the encrypted values. An example of a potentially guessable column would a three-digit credit card verification code.

Using Always Encrypted with Randomized encryption is more limited because the randomization means that the same value isn't always encrypted the same way. The only thing you can do with columns with Randomized encryption is to return them in your results. Deterministic encryption always encodes a value as the same string, which allows us to compare columns to a constant using equality and inequality operators, and to compare columns with each other to perform joins, grouping, and indexing.

Another thing to note is that the wizard is generating a column encryption key, which is the key that actually performs the data encryption. Each column being encrypted may have its own key, or as shown here, you can use the same key to encrypt both columns.

After identifying the columns you're encrypting, you can select **Next** and you'll see the **Master Key Configuration** screen:

In this screen, you create the column master key, which is used to encrypt the column encryption keys. You can supply your own key, if you're using T-SQL to encrypt the columns. This key must be stored in a key store such as the Windows Certificate Store, Azure Key Vault, or a hardware security module. The database engine never stores the column master key, and only contains the metadata about where it's stored. Not storing the master key protects data access from users who have full access to the database.

For the highest level of security, the key should be stored within a third party key store such as Azure Key Vault. Never generate the keys on the server hosting your database, as the key could potentially be extracted from memory on that server.

In the example below, the key is being stored in Azure Key Vault. On the next screen, the wizard will provide you the option to either finish the encryption process now, or to generate a PowerShell script. Once you complete the process, the data will appear as encrypted to anyone querying the data without the key.

In order to decrypt data from an Always Encrypted column, your application needs an Always Encrypted driver to connect to the database, followed by the actions below:

1. The application has access to the key store where the Always Encrypted keys are stored
2. The application then retrieves the data
3. Data that is written back to the database is encrypted at the client through the driver

In addition to the driver, the application's connection string needs to have the setting **Column Encryption Setting=enabled** provided. This setting will cause a metadata lookup to be made for each column that is used by the application.

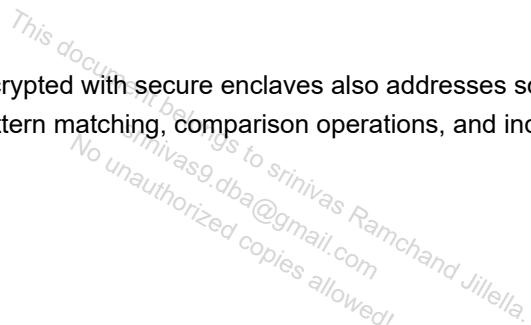
NOTE: To minimize metadata lookups, the application needs to update the *SqlCommandColumnEncryptionSetting* on the *SqlConnection* objects within the .NET application. These settings must be set for each database query that the application submits.

Secure enclaves

Always Encrypted supports a feature called secure enclaves, which allows more robust querying of encrypted data.

A secure enclave is a secured region of memory within the SQL Server process that acts as a trusted execution environment for processing encrypted data. This enclave appears as a black box to SQL Server, and it isn't possible to view any data or code, even with a debugger.

The image below shows the architecture of this process:



Always Encrypted with secure enclaves also addresses some of the limitations of Randomized encryption, which enables pattern matching, comparison operations, and indexing on columns using this encryption type.



Enable encrypted connections

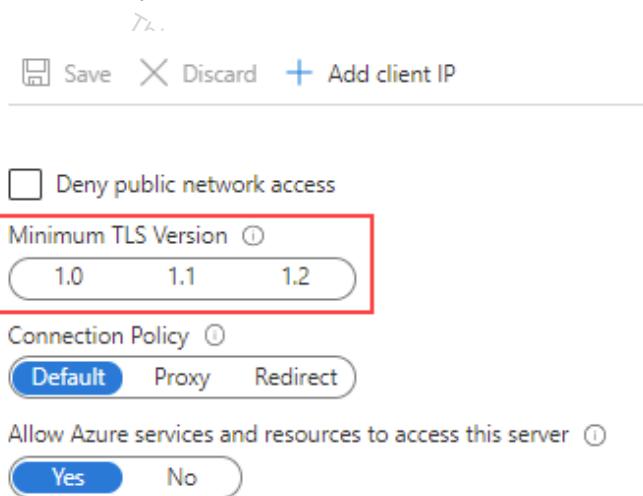
It is possible to encrypt data that is transmitted across a network between an instance of SQL Server and a client application with Transport Layer Security (TLS). The TLS encryption is performed within the protocol layer and is available to all supported SQL Server and Azure SQL database services.

Transport Layer Security (TLS)

Transport Layer Security (TLS) is a protocol for encrypting connections, and it increases the security of data being transmitted across networks between instances of SQL Server and applications.

Once a private certificate is issued by a certificate authority, and it has been assigned to a server running a SQL Server instance, the server can then use it to securely validate client requests. Such validation requires that the computer running the client application is configured to trust the certificate used by SQL Server.

For Azure SQL Database, it is possible to enforce a minimal TLS version at the server level. The TLS versions currently supported are **1.0**, **1.1**, and **1.2**. If the client application supports encrypted connection, we recommend setting it to the latest TLS version supported after testing the application. Choosing the latest TLS version is a good practice as it provides additional fixes for vulnerabilities found in older versions.



The minimal TLS version setting for a SQL Database is accessible from the **Firewalls and virtual networks** screen, as shown in the image above.

The following table explains the options available depending on whether client applications support encrypted connections.

Scenario	Option
TLS not supported	Leave the minimum TLS version setting at the default.
Latest TLS version supported	Set the minimum TLS version setting to 1.2.
Older TLS version supported	Set the minimum TLS version setting to 1.0 or 1.2. Evaluate your workloads for TLS 1.2 readiness and develop a migration plan.

Enabling TLS encryption offers privacy and data security for communications between instances of SQL Server and applications. However, when the traffic between SQL Server and a client application is encrypted with TLS, an extra network roundtrip and additional processing are required at connect time.

For Azure SQL Managed Instance, use `az sql mi update` Azure CLI command or `Set-AzSqlInstance` PowerShell cmdlet to configure the minimum TLS version.

NOTE: By default, Azure SQL Database does not require a specific minimum TLS version. Once you enforce a version of TLS, you can no longer revert to the default.

Certificates

You must run SQL Server Configuration Manager with a local administrator account in order to install certificates for use by SQL Server.

Furthermore, the certificate must satisfy the following conditions:

- The certificate must be located in the local computer certificate store or the current user certificate store.
- The SQL Server service account must have permission to access the certificate.
- The certificate must be within a valid period.

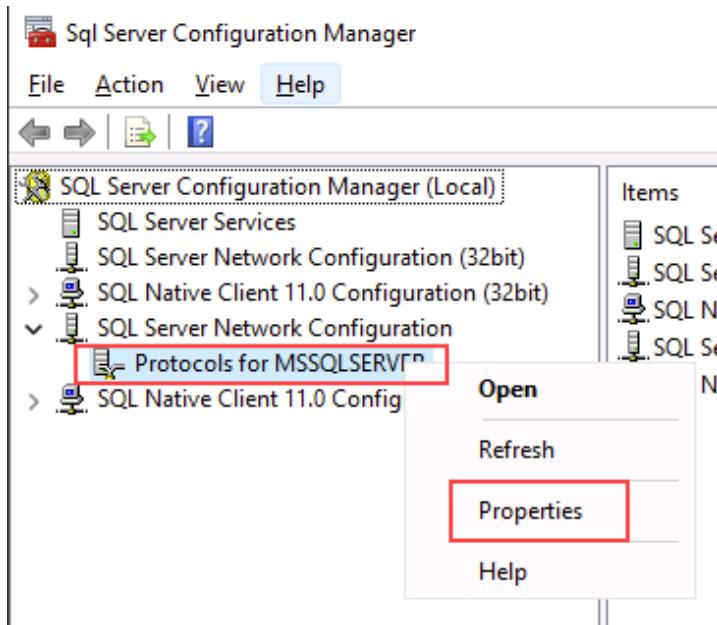
NOTE: If the correct access is not provided, restarting SQL Server service will fail.

For a complete list of requirements when installing a TLS certificate, see [Enable encrypted connections to the Database Engine](#).

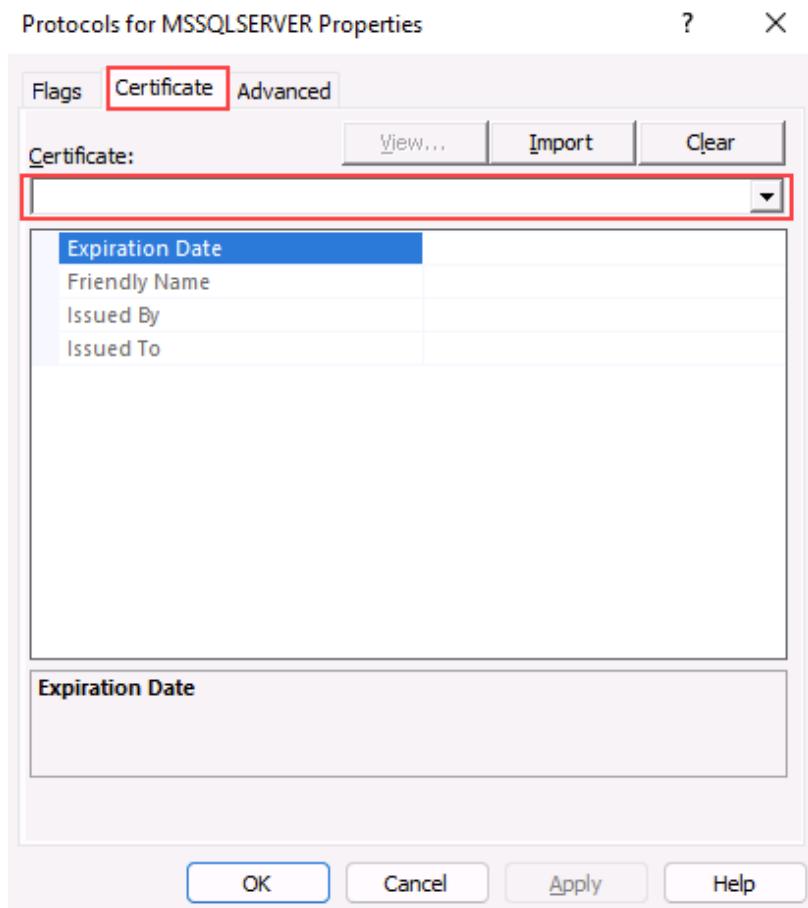
Configure SQL Server instance

You can configure a SQL Server instance to use encrypted connections by following these steps:

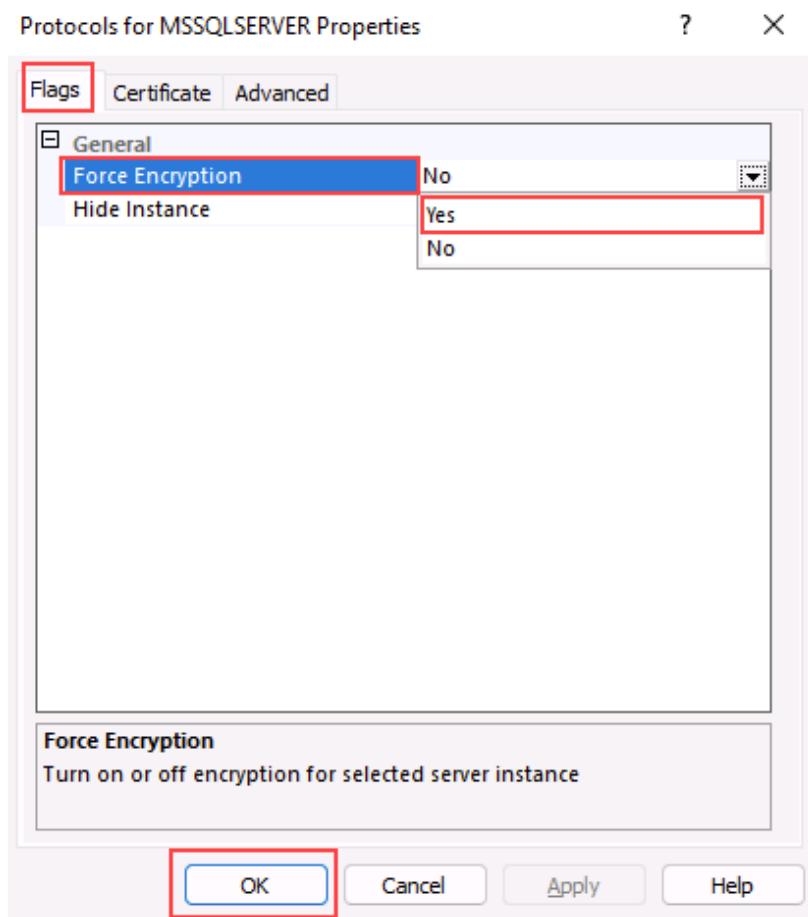
1. On the **SQL Server Configuration Manager**, expand **SQL Server Network Configuration**, right-click **Protocols for <server instance>**, and then select **Properties**.



2. From the **Protocols for <server instance> Properties** dialog box, select the **Certificate** tab, then select the certificate from the **Certificate** drop-down.



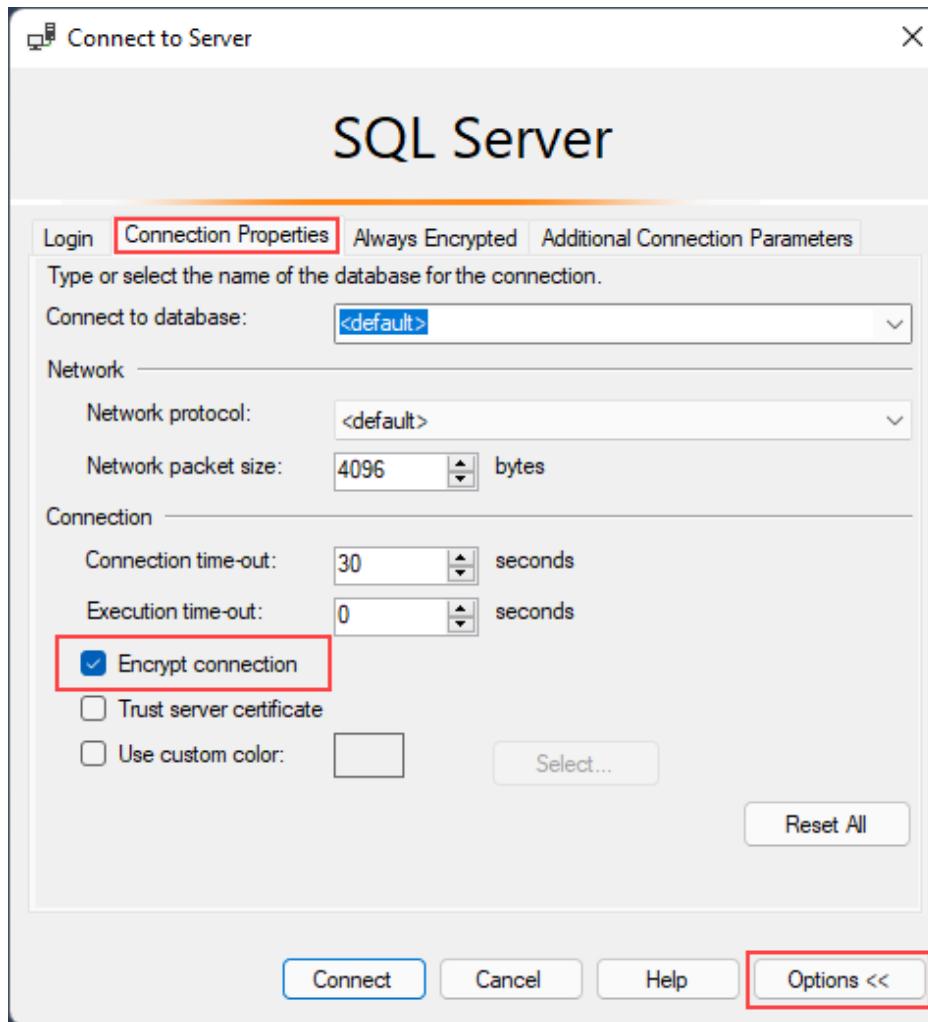
3. On the **Flags** tab, in the **ForceEncryption** property, select **Yes**, and then click **OK**.



4. Restart the SQL Server service.

Once the necessary configuration is in place, you can test the connection through SQL Server Management Studio:

1. In the **Connect to Server** dialog box, complete the connection information, and then click **Options**.
2. On the **Connection Properties** tab, click **Encrypt connection**, and then **Connect**.



All steps must have been executed correctly for you to be able to authenticate through SQL Server Management Studio using TLS.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Describe SQL injection

SQL injection is one of the most common methods used for data breaches. The core of the attack is that an SQL command is appended to the back end of a form field in the web or application front end (usually through a website), with the intent of breaking the original SQL Script and then executing the SQL script that was injected into the form field. This SQL injection most often happens when you have dynamically generated SQL within your client application. The core reason behind an SQL Injection attack comes down to poor coding practices both within the client application and within the database stored procedures. Many developers have learned better development practices, but SQL injection is still a significant problem due to both the number of legacy applications still being used and newer applications built by developers who didn't take SQL injection seriously while building the application.

As an example, assume that the front-end web application creates a dynamic SQL statement as shown below:

```
SELECT * FROM Orders WHERE OrderId=25
```

This T-SQL is created when the user goes to the sales order history portion of the company's website and enters 25 into the form field for the order ID number. However, suppose the user enters more than just an ID number, for example "25; DELETE FROM Orders;"

In that case, the query sent to your database would be as shown below:

```
SELECT * FROM Orders WHERE orderId=25; DELETE FROM Orders;
```

The way the query in the above example works is that the SQL database is told via the semicolon ";" that the statement has ended and that there is another statement that should be run. The database then processes the next statement as instructed, which would result in the deletion of all rows from the Orders table.

The initial SELECT query is run as normal without any errors being generated. However, when you look at the Orders table, you won't see any rows. The second query in the batch, which deletes all the rows, was also executed.

One technique used to prevent SQL injection attacks is to inspect the text of the parameters, or values entered into the form fields, looking for various keywords. However, this solution only provides minimum protection as there are many, many ways to force these attacks to work. Some of these injection techniques include passing in binary data, having the database engine convert the binary data back to a text string, and then executing the string. You can see a simple example of this problem by running the T-SQL code below.

```
DECLARE @v VARCHAR(255)

SELECT @v = cast(0x73705F68656C706462 AS VARCHAR(255))

EXEC (@v)
```

When data is being accepted from a user, either a customer or an employee, one good way to ensure that the value won't be used for an SQL injection attack is to validate that the data that was entered is of the expected data type. If a number is expected, the client application should ensure that there is in fact a number being returned. If a text string is expected, then ensure that the text string is of the correct length, and it does not contain any binary data within it. The client application should be able to validate all data being passed in from the user. Validation can

be done either by informing the user of the problem and allowing the user to correct the issue, or by crashing gracefully in such a way that an error is returned and no commands are sent to the database or the file system.

While fixing your application code should always be the priority, in some cases that may not be possible, so having Advanced Threat Protection can provide an additional layer of protection for your sensitive data.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Understand Azure Key Vault

Azure Key Vault is a tool used for storing and accessing secrets. Whether they be passwords, certificates or keys, Key Vault acts as a secure area for those secrets to be accessed in a secure fashion, typically programmatically. Key Vault data has its own RBAC policies, separate from the Azure subscription. This means someone who is in the role of subscription admin will not have access to the Key Vault unless explicitly granted.

SQL Server, either within an Azure Virtual Machine or on-premises, supports using Azure Key Vault to store certificates for features such as Transparent Data Encryption, Backup Encryption, or Always Encrypted. While this configuration is complex in an on-premises environment, it is easily managed when using SQL Server on Azure Virtual Machine, as shown in the image below.

The screenshot shows the Microsoft Azure portal interface for creating a new virtual machine. The top navigation bar includes 'Home', 'New', 'Marketplace', 'Get Started', 'SQL Server 2017 on Windows Server 2016', and 'Create a virtual machine'. On the left, there's a sidebar with various icons. The main page title is 'Create a virtual machine'. Below it, tabs include 'Basics', 'Disks', 'Networking', 'Management', 'Advanced', and 'SQL Server settings' (which is highlighted with a red box). The 'SQL AUTHENTICATION' section contains fields for 'SQL Authentication' (with 'Disable' and 'Enable' buttons) and 'Azure Key Vault integration' (with 'Disable' and 'Enable' buttons, where 'Enable' is highlighted with a red box). Other fields include 'Key Vault URL' (https://contosokeyvault.azure.net), 'Principal name', 'Principal secret' (redacted), and 'Credential name'.

In order to configure the Azure Key Vault integration, you need to set the Key Vault URL, the Principal name, the Principal secret, and the name of the credential. This task can be done at the virtual machine creation or to an existing VM.

Configuring SQL Server to connect to Azure Key Vault first requires creating a normal SQL Server Login within the instance. Next a Credential needs to be created and mapped to the login. For the identity of the credential, the name of the key vault should be used. For the secret of the credential, use the application ID from Azure Key Vault.

Once the credential is created, an asymmetric key can be created within the Azure Key Vault. An asymmetric key can then be created within the SQL Server database. The key in database can be mapped to the Azure Key Vault asymmetric key using the `CREATE ASYMMETRIC KEY` command with the `FROM PROVIDER` syntax. Once an asymmetric key is created within the database, that key can be used for TDE, or Backup Encryption or Always Encrypted.

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which security object is required in order to enable transparent data encryption?

- Credential
- Master Key
- User

Check Answers

Multiple choice

Which feature prevents members of the sysadmin role from viewing the values of data in a table?

- Always Encrypted
- Dynamic Data Masking
- Transparent Data Encryption

Check Answers

Multiple choice

Which TLS version offers the most secure option for encryption?

- TLS 1.1 version
- TLS 1.0 version
- TLS 1.2 version

Check Answers

Multiple choice

Which feature provides a private IP address for an Azure SQL Database?

- Network Endpoints
- Private Link
- Database Firewall

Check Answers

Multiple choice

Which technique can be used to create database firewall rules in Azure SQL Database?

- Running a PowerShell script
- Running an Azure CLI script
- Executing a T-SQL statement

Check Answers

Summary

This module explored the encryption options available within Microsoft SQL Server and Azure SQL. Each of the various platforms supports different database encryption options. You also explored these data encryption options and how to configure them.

Now that you've reviewed this module, you should be able to:

- Understand the difference between server and database firewall rules in Azure SQL Database
- Understand how Always Encrypted is used to protect data in transit
- Understand the role of Azure Key Vault in Transparent Data Encryption
- Explain how to enable encrypted connections

Introduction

This module explores the practices of setting compliance controls on the data that is stored within Azure SQL. We will also review important security features available on Azure to maintain compliance and improve visibility into any security risks.

Learning objectives

After taking this module, you will understand:

- How data should be classified
- Why server and database audit are important
- How to implement row level security and dynamic data masking
- Understand the usage of Microsoft Defender for SQL
- How Azure SQL Database Ledger works
- Explore Azure Purview supported capabilities

Explore data classification

Confidential data stored within a Microsoft SQL Server, Azure SQL Database or Azure SQL Managed Instance should be classified within the database. This classification allows users as well as other applications to know the sensitivity of the data that is being stored.

Data classification with the database is done on a column by column basis. It is possible for a single table to have some columns be public, some columns be confidential, and some columns be highly confidential.

Data classification was first introduced into SQL Server Management Studio and used extended properties of objects to store its data classification information. Starting with SQL Server 2019 this metadata is now stored in a catalog view called `sys.sensitivity_classifications`, and it is also supported by Azure SQL Database and Azure SQL Managed Instance.

The Azure portal provides a management pane for data classification of your Azure SQL Database as shown below. You can access this feature by selecting **Data Discovery and Classification**, which is in the **Security** section of the main blade for your Azure SQL Database.

Schema	Table	Column	Information type	Sensitivity label
SalesLT	Address	AddressLine1	Contact Info	Confidential
SalesLT	Address	AddressLine2	Contact Info	Confidential
SalesLT	Address	City	Contact Info	Confidential
SalesLT	Address	PostalCode	Contact Info	Confidential
SalesLT	Customer	FirstName	Name	Confidential - GDPR
SalesLT	Customer	LastName	Name	Confidential - GDPR
SalesLT	Customer	EmailAddress	Contact Info	Confidential

In both the Azure portal and SQL Server Management Studio, you can configure data classification. The classification engine scans your database and locates columns with names that indicate that the column could have sensitive information. For example, a column named `email` would be recommended by default as containing sensitive personal information.

The screenshot shows a list of 5 columns with classification recommendations. The columns are:

Schema	Table	Column	Information type	Sensitivity label
dbo	ErrorLog	UserName	Credentials	Confidential
SalesLT	CustomerAddress	AddressType	Contact Info	Confidential
SalesLT	SalesOrderHeader	AccountNumber	Financial	Confidential
SalesLT	SalesOrderHeader	CreditCardApprovalCode	Credit Card	Confidential
SalesLT	SalesOrderHeader	TaxAmt	Financial	Confidential

In the example above, there are five columns recommended for classification. The **Information Type** and **Sensitivity label** properties seem consistent with the column name and overall purpose. However, since the recommendations are based on the column name, a column named *column1* that contains email addresses would not be recommended as sensitive personal information.

Columns can also be classified using the sensitivity wizard in SQL Server Management Studio, or by using the **ADD SENSITIVITY CLASSIFICATION** T-SQL command as shown below:

```
ADD SENSITIVITY CLASSIFICATION TO
    [Application].[People].[EmailAddress]
WITH (LABEL='PII', INFORMATION_TYPE='Email')

GO
```

Classification of data allows you to easily identify the sensitivity of data within the database. Knowing what columns contain sensitive data allows for easier audits and allows you to more easily identify which columns are good choices for data encryption. Classification will allow other employees within the company to make better decisions on how to handle the data which is available within the database.

Customize classification taxonomy

Data Discovery & Classification is part of Microsoft Defender for Cloud. You can customize the taxonomy of sensitivity labels and define a set of classification rules specifically for your environment.

You can create and manage sensitivity labels as part of policy management by selecting **Data Discovery and Classification** of the main blade for your Azure SQL Database, and then **Configure**.

The screenshot shows the Data Discovery & Classification blade for the AdventureWorks database. The 'Classification' tab is selected. Key statistics shown are:

- Classified columns: 10
- Tables containing sensitive data: 2
- Unique information types: 3

On the **Information Protection** page you can define labels, rank them, and link them with a set of information types.

[Home](#) > [SQL databases](#) > [AdventureWorks \(dp300-lab-1922/AdventureWorks\)](#) >

Information Protection (preview)

X

[Save](#)[Discard](#)[Create label](#)[Manage information types](#)[Import/Export](#)[Learn more - Getting Started Guide](#)

Create and manage sensitivity labels

Drag labels to order in ascending sensitivity (least sensitive at the top)

[Configure](#)[Move up](#)[Move down](#)[Move to top](#)[Move to bottom](#)[Delete](#)

<input type="checkbox"/> Display name	State	Description
<input type="checkbox"/> Public	Enabled	...
<input type="checkbox"/> General	Enabled	...
<input type="checkbox"/> Confidential	Enabled	...
<input type="checkbox"/> Confidential - GDPR	Enabled	...
<input type="checkbox"/> Highly Confidential	Enabled	...
<input type="checkbox"/> Highly Confidential - GDPR	Enabled	...

[Create new label](#)

Once you define the patterns, they are added automatically to the discovery logic for identifying this type of data in your databases, and are immediately available.

NOTE: Only users with administrative rights on the organization's root management group can create and manage sensitivity labels.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Explore server and database audit

Azure SQL auditing tracks database events and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hubs. In addition, it enables you to maintain regulatory compliance, analyze activity patterns, and identify deviations that may indicate security violations.

You can define server-level and database-level policies. Server policies automatically cover new and existing databases in Azure.

- If server auditing is enabled, the database will be audited, regardless of the database auditing settings.
- In addition to enabling auditing on the server, you can also enable it on the database. This allows both audits to exist simultaneously; the server policy and the database policy.

It is best not to enable both server auditing and database auditing together, unless:

- A different *storage account*, *retention period* or *Log Analytics Workspace* is used for a specific database.
- An audit is needed for a specific database that differs from the rest on the server, such as different event types or categories.

For all the other cases, we recommend that you enable only server-level auditing and leave the database-level auditing disabled for all databases.

The default auditing policy for SQL Database includes the following set of action groups:

Action group	Definition
BATCH_COMPLETED_GROUP	Audits all the queries and stored procedures executed against the database.
SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP	This indicates that a principal succeed to log into the database.
FAILED_DATABASE_AUTHENTICATION_GROUP	This indicates that a principal failed to log into the database.

To enable auditing for all databases on an Azure SQL server, select **Auditing** in the **Security** section of the main blade for your server.

The screenshot shows the Azure portal interface for managing a SQL server named 'contoso-server'. In the left sidebar, under 'Security', the 'Auditing' option is selected and highlighted with a red box. The main content area displays the 'Essentials' blade for auditing. It includes sections for 'Active Directory admin', 'Microsoft Defender for SQL', 'Automatic tuning', and 'Auditing'. The 'Auditing' section is described as tracking database events and writing them to an audit log in Azure storage. The status for 'Auditing' is listed as 'NOT CONFIGURED'.

The **Auditing** page allows you to set the audit log destination and also choose whether to track Microsoft support engineer operations on the same log destination as Azure SQL Auditing or select a different one.

Save Discard Feedback

Azure SQL Auditing

Azure SQL Auditing tracks database events and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hub. [Learn more about Azure SQL](#)

[Auditing](#)

Enable Azure SQL Auditing

Audit log destination (choose at least one):

- Storage
- Log Analytics
- Event Hub

Auditing of Microsoft support operations

Auditing of Microsoft support operations tracks Microsoft support engineers' (DevOps) operations on your server and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hub. [Learn more about Auditing of Microsoft support operations](#)

Enable Auditing of Microsoft support operations

Use different audit log destinations

- Storage
- Log Analytics
- Event Hub

You can review the audit logs of Microsoft Support operations in your Log Analytics workspace by running the following query:

```
AzureDiagnostics
| where Category == "DevOpsOperationsAudit"
```

The auditing services for SQL Database and SQL Managed Instance are optimized for availability and performance. SQL Database and SQL Managed Instance may not record some audited events when there is a high rate of activity or high network load.

NOTE: Auditing on Read-Only replicas is automatically enabled.

Audit sensitive labels

When combined with data classification, you can also monitor access to sensitive data. Azure SQL Auditing has been enhanced to include a new field in the audit log called **data_sensitivity_information**.

By logging the sensitivity labels of the data returned by a query, this field provides an easier way to track access to classified columns.

name	action_name	class_type_desc	data_sensitivity_information
audit_event	BATCH COMPLETED	DATABASE	
audit_event	BATCH COMPLETED	DATABASE	
audit_event	BATCH COMPLETED	DATABASE	
audit_event	BATCH COMPLETED	DATABASE	<sensitivity_attributes max_rank="20" max_rank_desc="Medium"><sensitivity_attribute label="Confidential - GDPR" label_id="bf91e08cf4f0-478a-b016-25164b2a65ff" information_type="Name">
audit_event	BATCH COMPLETED	DATABASE	<sensitivity_attributes max_rank="20" max_rank_desc="Medium"><sensitivity_attribute label="Confidential - GDPR" label_id="bf91e08cf4f0-478a-b016-25164b2a65ff" information_type="Name">
audit_event	BATCH COMPLETED	DATABASE	<sensitivity_attributes max_rank="20" max_rank_desc="Medium"><sensitivity_attribute label="Confidential - GDPR" label_id="bf91e08cf4f0-478a-b016-25164b2a65ff" information_type="Name">
audit_event	BATCH COMPLETED	DATABASE	<sensitivity_attributes max_rank="20" max_rank_desc="Medium"><sensitivity_attribute label="Confidential - GDPR" label_id="bf91e08cf4f0-478a-b016-25164b2a65ff" information_type="SSN">

Auditing consists of tracking and recording events that occur in the database engine. Azure SQL auditing simplifies the configuration steps required to enable it, making it easier to track database activities for SQL Database and SQL Managed Instance.

Implement Dynamic Data Masking

Dynamic Data Masking works by obfuscating data in order to limit its exposure. Users who do not need to see sensitive data can view the column that contains the data, but not the actual data itself. Note that Dynamic Data Masking works at the presentation layer, and that unmasked data will always be visible by high privileged users.

Dynamic Data Masking has the advantage that it doesn't require many modifications to the application or database. You can configure it through the Azure portal, or using T-SQL as shown below:

```

ALTER TABLE [Application].[People] ALTER COLUMN [PhoneNumber] ADD MASKED WITH (FUNCTION = 'partial(0,"XXX-XXX-",4)')
ALTER TABLE [Application].[People] ALTER COLUMN [EmailAddress] ADD MASKED WITH (FUNCTION = 'email()')

EXECUTE AS USER = 'DDMDemo'

SELECT [PersonID]
      ,[FullName]
      ,[PhoneNumber]
      ,[EmailAddress]
  FROM [WideWorldImporters].[Application].[People]
 WHERE PhoneNumber IS NOT NULL
  
```

PersonID	FullName	PhoneNumber	EmailAddress
1	2 Kayla Woodcock	XXX-XXX-0102	kXXX@XXXX.com
2	3 Hudson Ondlow	XXX-XXX-0102	hXXX@XXXX.com
3	4 Isabella Rupp	XXX-XXX-0102	iXXX@XXXX.com
4	5 Eva Murden	XXX-XXX-0102	eXXX@XXXX.com
5	6 Sophia Hinton	XXX-XXX-0102	sXXX@XXXX.com
6	7 Amy Trell	XXX-XXX-0102	aXXX@XXXX.com
7	8 Anthony Grose	XXX-XXX-0102	aXXX@XXXX.com

In the above example, both the *PhoneNumber* and *EmailAddress* columns are hidden from *DDMDemo* user who only has *SELECT* permission on the table. The user is allowed to see the last four digits of the phone number as it's masked using a *partial* function that replaces all but the last four digits in the column. This masking is considered to be a custom function. In addition to T-SQL, if you are using Azure SQL Database, you can create dynamic masking rules in the Azure portal:

The screenshot shows the 'Add masking rule' blade in the Azure portal. The URL is [Home > AdventureWorks \(dp300-lab06-xyz/AdventureWorks\) | Dynamic Data Masking > Add masking rule](#).

Mask name: HumanResources_Employee_NationalID ...

Select what to mask:

- Schema:** HumanResources
- Table:** Employee
- Column:** NationalIDNumber (nvarchar)

Select how to mask:

Masking field format: Default value (0, xxxx, 01-01-1900)

Other options shown in the dropdown menu include:

- Default value (0, xxxx, 01-01-1900)
- Credit card value (xxxx-xxxx-xxxx-1234)
- Email (aXXX@XXXX.com)
- Number (random number range)
- Custom string (prefix [padding] suffix)

You can reach the screen to add a masking rule by navigating to your database in the Azure portal and selecting **Dynamic Data Masking** in the **Security** section of the main blade for your database.

Dynamic Data Masking supports the following masking patterns that can be used:

Masking function	Definition	T-SQL example
Default	Masks the data in the column without exposing any part of the values to the user. The user would see XXXX for string values, 0 for numbers, and 01.01.1900 for date values.	<pre>ALTER TABLE [Customer] ALTER COLUMN Address ADD MASKED WITH (FUNCTION = 'default')</pre>
Credit card	Masks all but the final four characters, allowing users to view the final four digits. This masking can be useful for customer service agents who need to view the last four digits of a credit card number but who do not need to see the entire number. The data is shown in the usual format of a credit card number XXXX-XXXX-XXXX-1234.	<pre>ALTER TABLE [Customer] ALTER COLUMN Address ADD MASKED WITH (FUNCTION = 'partial')</pre>
Email	Only the first letter and the trailing domain suffix are not masked; for example, "aXXX@XXXXXX.com"	<pre>ALTER TABLE [Customer] ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()')</pre>
Number	This masking format should be used on numeric columns. It shows a random number as the masked value instead of the actual value. With each query, a different number is displayed.	<pre>ALTER TABLE [Customer] ALTER COLUMN [Month] ADD MASKED WITH (FUNCTION = 'random(1,12)')</pre>
Custom string	This option allows text to be masked with any value, and to display a custom number of characters at either end of the masked value. If the length of the value being masked is equal to or less than the number of characters which the mask specifies are to be displayed, then only the masked characters are displayed.	<pre>ALTER TABLE [Customer] ALTER COLUMN [PhoneNumber] ADD MASKED WITH (FUNCTION = 'pathtext')</pre>

To enable users to retrieve unmasked data from the columns for which masking is defined, you need to explicitly grant `UNMASK` permission.

NOTE: It is possible to identify masked data using inference based on the results. If you are using data masking, you should also limit the ability of the user to run ad hoc queries. For that reason, it is highly recommended to use dynamic data masking in conjunction with other security features, such as auditing, encryption, row level security in order to better protect sensitive data.

Use case

Data masking is a simple and lightweight feature, and it is ideal for a number of scenarios, including:

- Mask data from application users who have no direct access to the database.
- Restricting private information for a group of users.

- Provide masked data to external vendors, where you need to protect sensitive information while still preserving the relationships among items in the data.
- Export a copy of your production database to a lower environment for development purposes with a user who doesn't have `UNMASK` permission. The export of the data will be in a masked format.

Import and export data

Copying data from a masked column to another table using `SELECT INTO` or `INSERT INTO` results in masked data in the target table.

When a user without `UNMASK` privilege runs SQL Server Import and Export, the exported data file will contain masked data, and the imported database will contain inactively masked data.

To learn more about how Dynamic Data Masking works, see [Dynamic Data Masking](#).

This document belongs to Srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to Srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to Srinivas Ramchand Jillella.

Implement Row Level security

Row-level security (RLS) doesn't use encryption and operates at the database level to restrict access to a table by using a security policy based on group membership or authorization context. This functionally is equivalent to a **WHERE** clause.

The security policy invokes an inline table-valued function to protect access to the rows in a table.

Depending on the attribute of a user, the predicate determines if that user has access to the relevant information. When you run a query against a table, the security policy applies the predicate function. Depending on the business requirements, RLS can be as simple as **WHERE CustomerId = 29** or as complex as required.

There are two types of security policies supported by row-level security:

- **Filter predicates** - restrict data access that violates the predicate.

Access	Definition
SELECT	Can't view rows that are filtered.
UPDATE	Can't update rows that are filtered.
DELETE	Can't delete rows that are filtered.
INSERT	Not applicable.

- **Block predicates** - restrict data changes that violate the predicate.

Access	Definition
AFTER INSERT	Prevents users from inserting rows with values that violate the predicate.
AFTER UPDATE	Prevents users from updating rows to values that violate the predicate.
BEFORE UPDATE	Prevents users from updating rows that currently violate the predicate.
BEFORE DELETE	Blocks delete operations if the row violates the predicate.

Because access control is configured and applied at the database level, application changes are minimal - if any. Also, users can directly have access to the tables and can query their own data.

Row-level security is implemented in three main steps:

1. Create the users or groups you want to isolate access.
2. Create the inline table-valued function that will filter the results based on the predicate defined.
3. Create a security policy for the table, assigning the function created above.

The T-SQL commands below demonstrate how to use RLS in a scenario where user access is segregated by tenant:

```
-- Create supporting objects for this example
CREATE TABLE [Sales] (SalesID INT,
    ProductID INT,
    TenantName NVARCHAR(10),
    OrderQtd INT,
    UnitPrice MONEY)
GO

INSERT INTO [Sales] VALUES (1, 3, 'Tenant1', 5, 10.00);
INSERT INTO [Sales] VALUES (2, 4, 'Tenant1', 2, 57.00);
INSERT INTO [Sales] VALUES (3, 7, 'Tenant1', 4, 23.00);
INSERT INTO [Sales] VALUES (4, 2, 'Tenant2', 2, 91.00);
INSERT INTO [Sales] VALUES (5, 9, 'Tenant3', 5, 80.00);
INSERT INTO [Sales] VALUES (6, 1, 'Tenant3', 5, 35.00);
INSERT INTO [Sales] VALUES (7, 3, 'Tenant4', 8, 11.00);

-- View all the rows in the table
SELECT * FROM Sales;
```

Next, create the users and grant them access to the *Sales* table. In this example, each user is responsible for a specific tenant. The *TenantAdmin* user has access to see data from all tenants.

```
CREATE USER [TenantAdmin] WITH PASSWORD = '<strong password>'
GO
CREATE USER [Tenant1] WITH PASSWORD = '<strong password>'
GO
CREATE USER [Tenant2] WITH PASSWORD = '<strong password>'
GO
CREATE USER [Tenant3] WITH PASSWORD = '<strong password>'
GO
CREATE USER [Tenant4] WITH PASSWORD = '<strong password>'
GO

GRANT SELECT ON [Sales] TO [TenantAdmin]
GO
GRANT SELECT ON [Sales] TO [Tenant1]
GO
GRANT SELECT ON [Sales] TO [Tenant2]
GO
GRANT SELECT ON [Sales] TO [Tenant3]
GO
GRANT SELECT ON [Sales] TO [Tenant4]
GO
```

Next, we'll create a new schema, an inline table-valued function, and grant user access to the new function. The `WHERE @TenantName = USER_NAME() OR USER_NAME() = 'TenantAdmin'` predicate evaluates if the user name executing the query matches the *TenantName* column values.

```
CREATE SCHEMA sec;
GO
```

```
--Create the filter predicate

CREATE FUNCTION sec.tvf_SecurityPredicatebyTenant(@TenantName AS NVARCHAR(10))
    RETURNS TABLE
WITH SCHEMABINDING
AS
    RETURN      SELECT 1 AS result
                WHERE @TenantName = USER_NAME() OR USER_NAME() = 'TenantAdmin';
GO

--Grant users access to inline table-valued function

GRANT SELECT ON sec.tvf_SecurityPredicatebyTenant TO [TenantAdmin]
GO
GRANT SELECT ON sec.tvf_SecurityPredicatebyTenant TO [Tenant1]
GO
GRANT SELECT ON sec.tvf_SecurityPredicatebyTenant TO [Tenant2]
GO
GRANT SELECT ON sec.tvf_SecurityPredicatebyTenant TO [Tenant3]
GO
GRANT SELECT ON sec.tvf_SecurityPredicatebyTenant TO [Tenant4]
GO

--Create security policy and add the filter predicate
CREATE SECURITY POLICY sec.SalesPolicy
ADD FILTER PREDICATE sec.tvf_SecurityPredicatebyTenant(TenantName) ON [dbo].[Sales]
WITH (STATE = ON);
GO
```

At this point, we're ready to test the access:

```
EXECUTE AS USER = 'TenantAdmin';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant1';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant2';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant3';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant4';
SELECT * FROM dbo.Sales;
REVERT;
```

The *TenantAdmin* user should see all the rows. The *Tenant1*, *Tenant2*, *Tenant3* and *Tenant4* users should only see their own rows.

If you alter the security policy with `WITH (STATE = OFF);`, you'll notice that users will see all the rows.

```

ALTER SECURITY POLICY sec.SalesPolicy WITH (STATE = OFF);
GO

EXECUTE AS USER = 'Tenant1';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant2';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant3';
SELECT * FROM dbo.Sales;
REVERT;

EXECUTE AS USER = 'Tenant4';
SELECT * FROM dbo.Sales;
REVERT;

```

The screenshot shows the results pane with the following data:

	SalesID	ProductID	TenantName	OrderQtd	Unit Price
1	1	3	Tenant1	5	10.00
2	2	4	Tenant1	2	57.00
3	3	7	Tenant1	4	23.00
4	4	2	Tenant2	2	91.00
5	5	9	Tenant3	5	80.00
6	6	1	Tenant3	5	35.00
7	7	3	Tenant4	8	11.00

NOTE: There is a risk of information leakage if an attacker writes a query with a specially crafted `WHERE` clause and, for example, a divide-by-zero error, to force an exception if the `WHERE` condition is true. This is known as a *side-channel attack*. It is wise to limit the ability of users to run ad hoc queries when using row-level security.

Use case

Row-level security is ideal for many scenarios, including:

- When you need to isolate departmental access at the row level.
- When you need to restrict customers' data access to only the data relevant to their company.
- When you need to restrict access for compliance purposes.

Best practice

Here are a few best practices to consider when implementing RLS:

- It's recommended to create a separate schema for predicate functions, and security policies.
- Whenever possible, avoid type conversions in predicate functions.

- To maximize performance, avoid using excessive table joins and recursion in predicate functions.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Understand Microsoft Defender for SQL

Microsoft Defender for SQL offers a suite of protections for Azure SQL Database and Azure SQL Managed Instance as part of the advanced SQL security features, including SQL vulnerability assessment and Advanced Threat Protection.

SQL vulnerability assessment

SQL vulnerability assessment is a service that uses a knowledge base of security rules to flag items that don't comply when they're scanned. It checks your database for security best practices, and providing visibility into your security state, such as misconfigurations, excessive permissions, and exposure of sensitive data.

To see recommendations for SQL Database and SQL Managed Instance, you must enable Microsoft Defender for SQL at the subscription level (recommended). You also need to provide a storage account. Alternatively, you can choose to receive emails with a summary of the scan results.

Server settings ...

dp300-server

[Save](#) [Discard](#) [Feedback](#)

MICROSOFT DEFENDER FOR SQL

[ON](#)[OFF](#)

Microsoft Defender for SQL costs 15 USD/server/month. It includes Vulnerability Assessment and Advanced Threat Protection. We invite you to a trial period for the first 30 days, without charge.

VULNERABILITY ASSESSMENT SETTINGS

Subscription

Contoso Subscription

[Select Subscription](#)

Storage account

contosologs

[Select Storage account](#)

Periodic recurring scans

[ON](#)[OFF](#)

Scans will be triggered automatically once a week. In most cases, it will be on the day Vulnerability Assessment has been enabled and saved. A scan result summary will be sent to the email addresses you provide.

Send scan reports to

myteam@contoso.com

 Also send email notification to admins and subscription owners

ADVANCED THREAT PROTECTION SETTINGS

Advanced Threat Protection for SQL alerts emails are sent by Defender for Cloud.

[Add your contact details to the subscription's email settings in Defender for Cloud.](#)

[Enable Auditing for better threats investigation experience](#)

The vulnerability assessment feature can detect potential risks in your environment, and help you enhance database security. It also provides insight into your security state and actionable steps to resolve security alerts.

To learn more about SQL vulnerability assessment, see [SQL vulnerability assessment helps you identify database vulnerabilities](#).

Advanced Threat Protection

Advanced Threat Protection monitors the database connections and the queries that are executed in order to detect potentially harmful activities. You can manage and access Advanced Threat Protection via the central Microsoft Defender for SQL portal.

The following threats are supported by Advanced Threat Protection:

Alerts	Definition
Vulnerability to SQL injection	This alert looks for T-SQL code coming into your database that may be vulnerable to SQL injection attacks. An example would be a stored procedure call that didn't sanitize user inputs.
Potential SQL injection	This alert is triggered when an attacker is actively attempting to execute a SQL injection attack.
Access from unusual location	This alert is triggered when a user logs in from an unusual geographic location.
Access from unusual Azure data center	This alert is looking for attacks from an Azure data center that isn't normally accessed.
Access from unfamiliar principal	This alert is raised when a user or applications log on to a database that they haven't previously accessed.
Access from a potentially harmful application	This alert detects common tools that are used to attack databases.
Brute force SQL credentials	This alert is triggered when there a high number of log in failures with different credentials.

To get maximum benefit out of it you'll want to enable auditing on your databases. Although it isn't required, enabling auditing will allow for deeper investigation into the source of the problem if Advanced Threat Protection detects an anomaly.

The following audit action groups are recommended:

- **BATCH_COMPLETED_GROUP**
- **SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP**
- **FAILED_DATABASE_AUTHENTICATION_GROUP**

How to enable Microsoft Defender for SQL

You must belong to either the SQL security manager role, or one of the database or server admin role to manage Microsoft Defender for SQL settings.

Enable Microsoft Defender for SQL for SQL Database or SQL Managed Instance from the server main blade by selecting **Microsoft Defender for Cloud**, and then the **(Configure)** link.

Home > AdventureWorks (dp300-server/AdventureWorks)

AdventureWorks (dp300-server/AdventureWorks) | Microsoft Defender for Cloud

Visit Microsoft Defender for Cloud to manage security across your virtual networks, data, apps, and more

Recommendations: 0 | Security alerts: 0 | Findings: 0 | Microsoft Defender for SQL Enabled at the subscription-level (Configure)

Learn more: About Microsoft Defender for Cloud | About Microsoft Defender for SQL

Recommendations

Defender for Cloud continuously monitors the configuration of your SQL Servers to identify potential security vulnerabilities and recommends actions to mitigate them.

No recommendations to display

There are no security recommendations for this resource

[View all recommendations in Defender for Cloud](#)

Security incidents and alerts

Defender for Cloud uses advanced analytics and global threat intelligence to alert you to malicious activity. Alerts displayed below are from the past 21 days.

On the **Server settings** page, make sure the **MICROSOFT DEFENDER FOR SQL** property is set to **ON**.

Server settings

dp300-server

Save Discard Feedback

MICROSOFT DEFENDER FOR SQL

ON **OFF**

i Microsoft Defender for SQL costs 15 USD/server/month. It includes Vulnerability Assessment and Advanced Threat Protection. We invite you to a trial period for the first 30 days, without charge.

Once Microsoft Defender for SQL is enabled, you can view recommendations by selecting **Microsoft Defender for Cloud** on the server blade.

dp300-server | Microsoft Defender for Cloud

Visit Microsoft Defender for Cloud to manage security across your virtual networks, data, apps, and more

Recommendations Security alerts Findings Microsoft Defender for SQL: Enabled at the subscription-level (Configure) Learn more

4 1 0 --

Recommendations

Defender for Cloud continuously monitors the configuration of your SQL Servers to identify potential security vulnerabilities and recommends actions to mitigate them.

Description	Severity
Auditing on SQL server should be enabled	Low
Private endpoint connections on Azure SQL Database should be enabled	Medium
Public network access on Azure SQL Database should be disabled	Medium
SQL servers should have vulnerability assessment configured	High

[View additional recommendations in Defender for Cloud >](#)

Microsoft Defender for SQL provides advanced built-in features to identify and handle threats to the database without the need to be a security expert.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

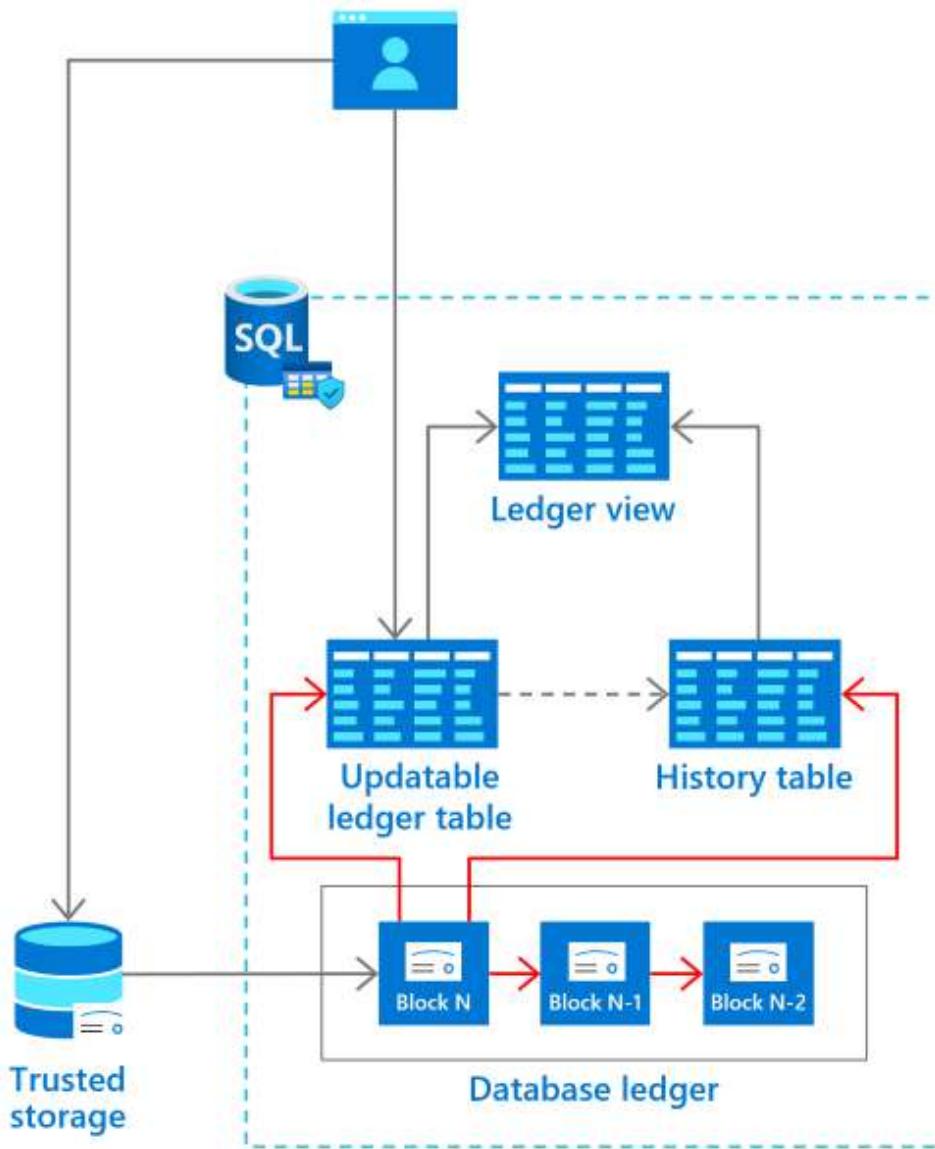
Explore Azure SQL Database Ledger

The ledger feature of Azure SQL Database provides tamper-evidence capabilities in your database. You can cryptographically attest to other parties, such as auditors or other business parties, that your data hasn't been tampered with.

How it works

Cryptography and blockchain have begun to appear in far reaching areas of technology with varying degrees of success. One place where it has proved useful and beneficial is in being used as the technology behind the Azure SQL Database Ledger feature. The Ledger feature provides tamper-evidence capabilities in your database. Using the Ledger feature of the Azure SQL Database, you can provide concrete proof to auditors, business partners or any interested parties what data has been changed or tampered with.

A traditional ledger is defined as a collection of accounts of a particular type and that's exactly what the Azure SQL Database Ledger feature provides in your environment. It provides transparent protection of your data from bad actors including but not limited to attackers or even database or cloud administrators. It provides guarantees of cryptographic data integrity while maintaining the power, flexibility, and performance of Azure SQL Database.



Each transaction that the database receives is cryptographically hashed (SHA-256). The function cryptographically links all transactions together, like a blockchain.

Components

Azure SQL Database Ledger function currently exists for tables in two forms: The Updatable Ledger Tables and the Append-only Ledger Tables.

Updatable ledger tables

Updatable ledger table can be used for applications that issue updates and deletes and inserts. It works well for system of record applications and transactional systems where matter of fact record keeping and auditing is required and happen. The updatable ledger tables track history of changes to any rows and uses the built-in system versioning to create a history table that stores the previous version of the row for full history is kept for any updates or deletes.

Append-only ledger tables

Append-only ledger tables work well with insert only applications such as an accounting system, which still needs auditing or security information and event management (SIEM) applications. The append-only ledger table blocks all updates and deletes at the API level so not only does it provide certainty, it aides in management.

Benefits

The Azure SQL Database Ledger feature provides multiple benefits:

Ease Audits – Audits are frequently enacted to ensure that proper security controls are in place to reduce potential attacks, backup and restore practices are as required, and thorough disaster recovery procedures are in place. Ledger provides documented proof that your data hasn't been altered in an auditing process.

Increased trust – Ledger also can help establish trust between multiple-party business processes without the complexity and performance implications that network consensus can introduce.

Data integrity – Querying the data on a blockchain network without sacrificing performance can be a serious challenge. Ledger provides data integrity for off-chain storage of blockchain networks, which helps ensure complete data trust through the entire system.

Create a SQL database with ledger enabled

You can easily enable the ledger feature by following the steps below:

1. Browse to the **Select SQL Deployment option** page on Azure portal.
2. Under **SQL databases**, leave **Resource type** set to **Single database**, and select **Create**.

Home > Azure SQL >

Select SQL deployment option ...

Microsoft

X



How do you plan to use the service?

SQL databases
Best for modern cloud applications. Hyperscale and serverless options are available.
Resource type: Single database
Create Show details

SQL managed instances
Best for most migrations to the cloud. Lift-and-shift ready.
Resource type: Single instance
Create Show details

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

3. On the **Basics** tab of the **Create SQL Database** form, under Project details, select the Azure subscription you want to use.
4. For **Resource group**, select **Create new**, enter *myResourceGroup*, and select **OK**.
5. For **Database name**, enter *demo*.
6. For **Server**, select **Create new**. Fill out the New server form with the following values, and select **OK**:
 - **Server name:** <enter a unique server name>
 - **Server admin login:** *azureuser*
 - **Password:** <enter a strong password>
 - **Location:** Select the same region where your resource group was created
 - **Allow Azure services to access this server:** Select this option to enable access to digest storage.
7. Leave **Want to use SQL elastic pool** set to **No**.
8. Under **Compute + storage**, select **Configure database**.
9. Select **Serverless**, and then select **Apply**.

Home > Azure SQL > Select SQL deployment option > Create SQL Database >

Configure ...

X



Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. Learn more

Service tier

General Purpose (Scalable compute and storage options)

Compare service tiers

Compute tier

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

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

Compute Hardware

Select the hardware configuration based on your workload requirements. Availability of compute optimized, memory optimized, and confidential computing hardware depends on the region, service tier, and compute tier.

Hardware Configuration

Gen5
up to 40 vCores, up to 120 GB memory

Apply

Cost summary

Gen5 - General Purpose (GP_S_Gen5_1)
Cost per GB (in USD)
Max storage selected (in GB)

ESTIMATED STORAGE COST / MONTH
COMPUTE COST / VCORE / SECOND

NOTES

10. On the **Networking** tab, for **Connectivity method**, select **Public endpoint**.

11. For **Firewall rules**, set **Add current client IP address** to **Yes**. Leave **Allow Azure services and resources to access this server** set to **No**.

12. Select **Next: Security** at the bottom of the page.

Create SQL Database

Microsoft

Basics Networking Security Additional settings Tags Review + create

Configure network access and connectivity for your server. The configuration selected below will apply to the selected server 'boogleserver' and all databases it manages. [Learn more](#)

Network connectivity

Choose an option for configuring connectivity to your server via public endpoint or private endpoint. Choosing no access creates with defaults and you can configure connection method after server creation. [Learn more](#)

Connectivity method *

- No access
 Public endpoint
 Private endpoint

Firewall rules

Setting 'Allow Azure services and resources to access this server' to Yes allows communications from all resources inside the Azure boundary, that may or may not be part of your subscription. [Learn more](#)

Setting 'Add current client IP address' to Yes will add an entry for your client IP address to the server firewall.

Allow Azure services and resources to access this server *

Add current client IP address *

13. On the **Security** tab, in the Ledger section, select the **Configure ledger** link.

Create SQL Database

Microsoft

...

[Basics](#) [Networking](#) [Security](#) [Additional settings](#) [Tags](#) [Review + create](#)

Azure Defender for SQL

Protect your data using Azure Defender for SQL, a unified security package including vulnerability assessment and advanced threat protection for your server. [Learn more](#)

Enable Azure Defender for SQL *

Start free trial

Not now

Azure Defender for SQL will automatically create a new storage account for saving vulnerability assessments. If a storage account was previously created for this purpose, it will be used instead. Azure storage prices will apply.

Ledger (preview)

Ledger cryptographically verifies the integrity of your data and detects any tampering that might have occurred. [Learn more](#)

Ledger (preview)

Not configured

[Configure ledger](#)

14. On the Configure ledger pane, in the **Ledger** section, select the **Enable for all future tables in this database** checkbox.
NOTE: This setting ensures that all future tables in the database will be ledger tables. For this reason, all data in the database will show any evidence of tampering. By default, new tables will be created as updatable ledger tables, even if you don't specify `LEDGER = ON` in the `CREATE TABLE` statement. You can also leave this option unselected. You're then required to enable ledger functionality on a per-table basis when you create new tables by using Transact-SQL.
15. In the **Digest Storage** section, **Enable automatic digest storage** is automatically selected. Then, a new Azure Storage account and container where your digests are stored is created.
16. Select **Apply**.

Configure ledger (preview)

[Create SQL Database](#)

i Azure SQL Ledger and Azure Confidential Ledger are each currently in preview. By using this preview feature, you confirm that you agree that your use of this feature is subject to the preview terms in the agreement under which you obtained Microsoft Azure Services [See preview terms](#)

Ledger (preview)

Enabling ledger functionality will make all tables in your database ledger tables that can be updated. This option cannot be changed after you create your database. If you do not select this option now, you can create ledger tables that can be updated or only appended to when creating new tables using T-SQL. After enabling ledger functionality for a table, you cannot disable this option [Learn more](#)

Enable for all future tables in this database

Digest Storage

If you want ledger to generate digests automatically and store them for your verification later, you need to configure an Azure Storage account or Azure Confidential Ledger. Alternatively, you can manually generate digests and store them in your own secure location [Learn more](#)

Enable automatic digest storage

Storage type [\(i\)](#)

Azure Storage

Azure Confidential Ledger (preview)

Storage account * [\(i\)](#)

(new) sqldbdigeststorage

[create new](#)

Storage container [\(i\)](#)

(new) preDefinedStorageContainer

Apply

17. Select **Review + create** at the bottom of the page.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Create SQL Database

Microsoft

[Basics](#) [Networking](#) [Security](#) [Additional settings](#) [Tags](#) [Review + create](#)

Azure Defender for SQL

Protect your data using Azure Defender for SQL, a unified security package including vulnerability assessment and advanced threat protection for your server. [Learn more](#)

Enable Azure Defender for SQL * [?](#)

Start free trial

Not now

Azure Defender for SQL will automatically create a new storage account for saving vulnerability assessments. If a storage account was previously created for this purpose, it will be used instead. Azure storage prices will apply.

Ledger (preview)

Ledger cryptographically verifies the integrity of your data and detects any tampering that might have occurred. [Learn more](#)

Ledger (preview) [?](#)

Configured

[Configure ledger](#)

[Review + create](#)

[< Previous](#)

[Next : Additional settings >](#)

18. On the **Review + create** page, after your review, select **Create**.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

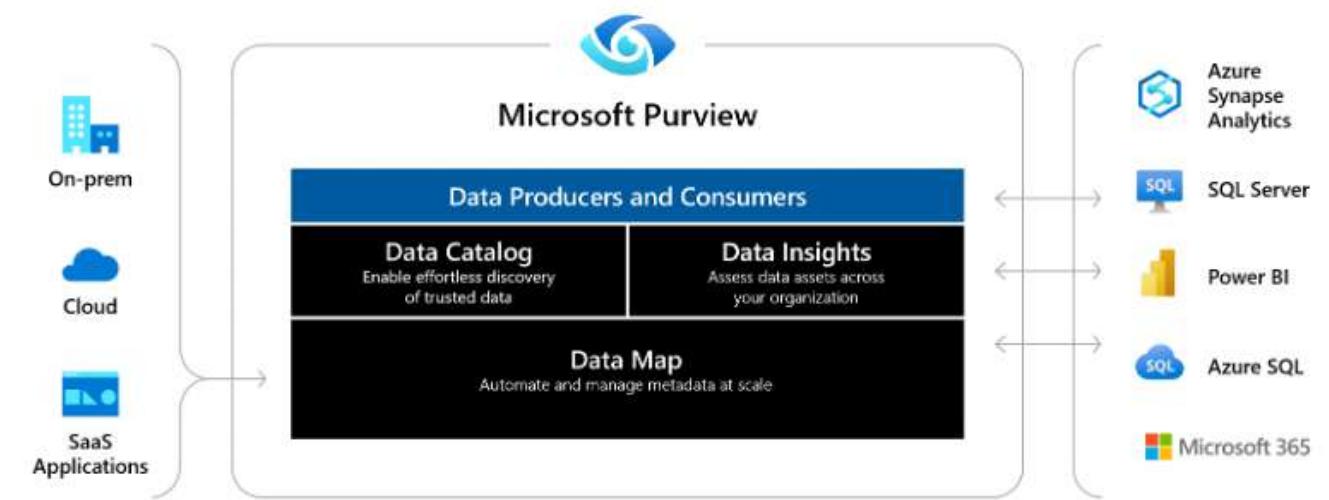
This document bel...

Implement Azure Purview

Microsoft Purview is a unified data governance service that helps you manage and govern your on-premises, multi-cloud, and software-as-a-service (SaaS) data. Create a holistic, up-to-date map of your data landscape with automated data discovery, sensitive data classification, and end-to-end data lineage. Enable data curators to manage and secure your data estate. Empower data consumers to find valuable, trustworthy data.

How it works

Microsoft Purview automates data discovery by providing data scanning and classification as a service for assets across your data estate. Metadata and descriptions of discovered data assets are integrated into a holistic map of your data estate. Atop this map, there are purpose-built apps that create environments for data discovery, access management, and insights about your data landscape.



Supported capabilities

Understanding the location and movement of sensitive data across the entire data domain is one of the main features of Azure Purview for Azure SQL Database.

Create a unified map of data across the entire data domain

Azure Purview helps you lay the foundation for effective data management, including the following capabilities:

- Automate and manage hybrid resource metadata.
- Classify data using integrated and custom classifications and information protection sensitivity labels.
- Ensure consistent labeling of sensitive data across SQL Server, Azure, Microsoft 365, and Power BI.
- Easily integrate all your data systems using Apache Atlas APIs.

The screenshot shows the Microsoft Purview Data Catalog interface. At the top, there's a navigation bar with 'Microsoft Azure' and 'Purview' followed by 'Adatum Corp'. A search bar contains the text 'Revenue'. On the right, there's a user icon for 'contoso@contoso.com' and a 'MICROSOFT' logo. Below the navigation, a sidebar on the left lists 'Sources', 'Home', 'Collections', 'Data dictionary', and 'Map view'. The main area displays five collections: 'NorthAmericaDataCenter', 'EuropeDataCenter', 'AzureAndBINorthAmerica', 'AmazonNorthAmerica', and 'AzureEurope'. Each collection has a 'View details' button. Underneath these collections are several data sources, each with its own icon, name, provider, and a 'View details' button. Some sources are grouped under specific collections.

Collection	Source	Type	View details	
NorthAmericaDataCenter	OnPremSQLServer-Fin...	SQL Server	View details	
	Teradata-FinanceData	Teradata (Preview)	View details	
	HiveMetastore	Hive Metastore (Preview)	View details	
	FinanceSQLServer	SQL Server	View details	
	Teradata	Teradata (Preview)	View details	
EuropeDataCenter	SAP-S4HANA-Procurement...	SAP S/4Hana (Preview)	View details	
	SAP-ECC-SalesData	SAP ECC (Preview)	View details	
	SAP-S4HANA	SAP S/4Hana (Preview)	View details	
	SAP-ECC	SAP ECC (Preview)	View details	
	OnPremSQLServer	SQL Server	View details	
AzureAndBINorthAmerica	AzureDataLakeStorage-...	Azure Data Lake Storage Gen2	View details	
	AzureBlobStorage	Azure Blob Storage	View details	
	AzureSQLDB-SalesInvoi...	Azure SQL Database	View details	
	RevenuePBIDashboards	Power BI	View details	
	WebLogs	Azure File	View details	
AmazonNorthAmerica	AmazonS3-HRData	Amazon S3	View details	
	AWSS3	Amazon S3	View details	
	AzureSqlManagedInsta...	Azure SQL Database Managed Instances	View details	
	Map view			
	AzureEurope	Azure Data Lake Storage Gen2	View details	

Make data easy to find

Make data easy to find using familiar business and technical search terms, including the following capabilities:

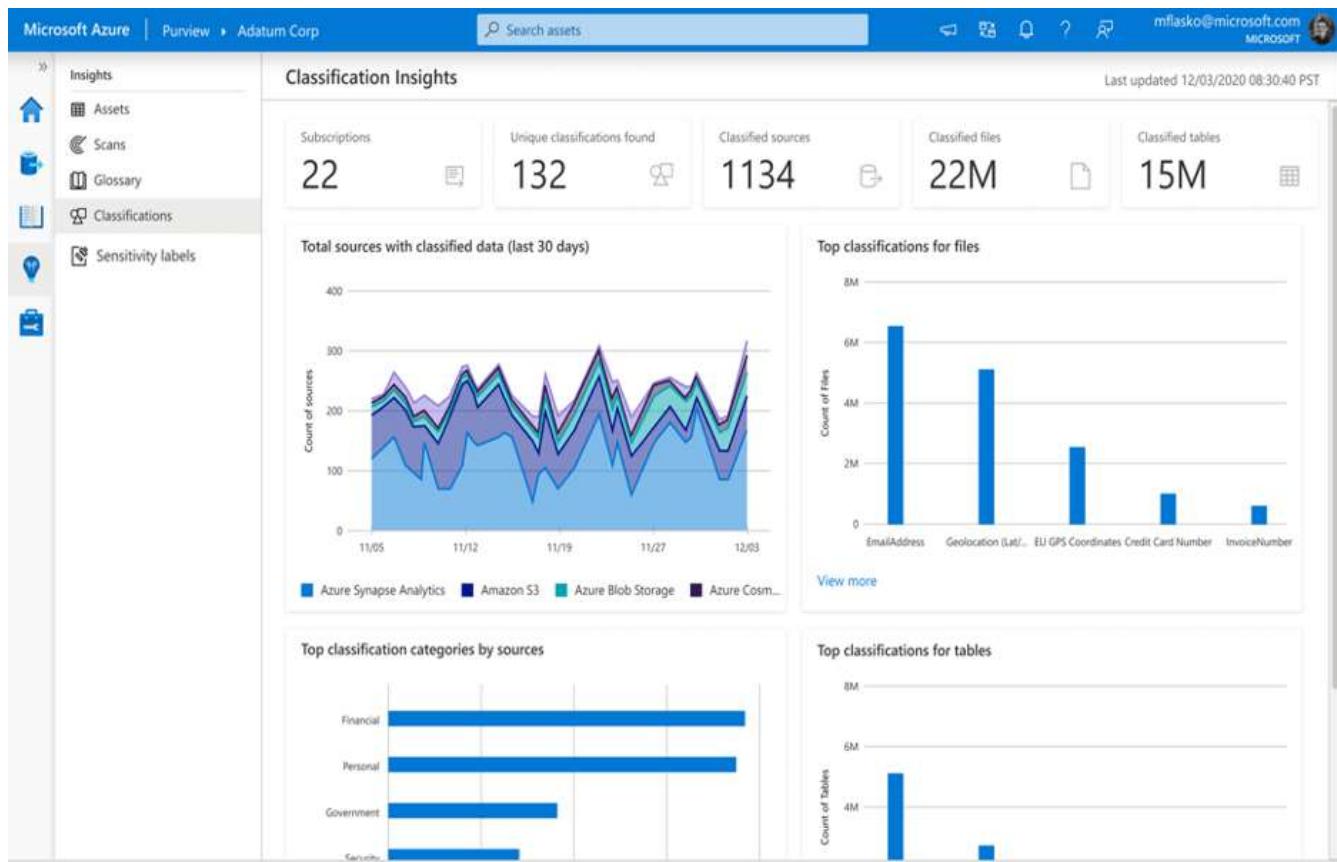
- Ensure optimal business value for your data users' data with Microsoft Purview Data Catalog.
- Eliminate the need for data dictionaries in Excel with a business-level business dictionary.
- Gain insight into the origin of your data with interactive visualization of data origin.
- Provide data scientists, engineers, and analysts with the data they need for BI, analytics, AI, and machine learning.

Get an overview of sensitive data

Microsoft Purview provides a comprehensive view of your data management operations with Data Insights (in preview), including the following capabilities:

- View your entire data domain and its distribution by asset dimension, such as source type, classification, and file size.
- Receive status updates on the number of scans that passed, failed, or canceled.
- Get key insights to add or redistribute glossary terms for better search results.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*



Requirements

Before you get started with Microsoft Purview, ensure the following requirements are met:

- Access to Microsoft Azure with a development or production subscription.
- Ability to create Azure resources including Microsoft Purview.
- Access to data sources such as Azure Data Lake Storage or Azure SQL in test, development, or production environments.
 - For Data Lake Storage, the required role to scan is **Reader**.
 - For Azure SQL, the identity must be able to query tables for sampling of classifications.
- Access to Microsoft Defender for Cloud or ability to collaborate with Defender for Cloud Admin for data labeling.
- An active Microsoft Purview account.
- You'll need to be a **Data Source Administrator** and **Data Reader** to register a source and manage it in the Microsoft Purview governance portal.

Security considerations

Let's review some important security capabilities when scanning a SQL Database using Microsoft Purview.

Firewall settings

If your database server has a firewall enabled, you'll need to update the firewall to allow access in one of two ways:

- **Allow Azure connections through the firewall** – A straightforward option to route traffic through Azure networking, without needing to manage virtual machines.
- **Install a self-hosted integration runtime** – Install a self-hosted integration runtime on a machine in your network and give it access through the firewall. If you have a private VNet set up within Azure, or have any other closed network set up, using a self-hosted integration runtime on a machine within that network will allow you to fully manage traffic flow and utilize your existing network.
- **Use a managed virtual network** – You can use the Azure integration runtime in a closed network by setting up a managed virtual network using your Microsoft Purview account to connect to Azure SQL.

Authentication

To scan your data source, you'll need to configure an authentication method in the Azure SQL Database. The following authentication options are supported when preparing for a scan:

- **System-assigned managed identity (recommended)** – This is an identity associated directly with your Microsoft Purview account that allows you to authenticate directly with other Azure resources without needing to manage a go-between user or credential set. The system-assigned managed identity is created when your Microsoft Purview resource is created, is managed by Azure, and uses your Microsoft Purview account's name. The system-assigned managed identity can't currently be used with a self-hosted integration runtime for Azure SQL.
- **User-assigned managed identity (preview)** – Similar to system-assigned managed identity, a user-assigned managed identity is a credential resource that allows Microsoft Purview to authenticate against Azure Active Directory. The user-assigned managed by users in Azure, rather than by Azure itself, which gives you more control over security. The user-assigned managed identity can't currently be used with a self-hosted integration runtime for Azure SQL. For more information, see our guide for user-assigned managed identities.
- **Service Principal** – A service principal is an application that can be assigned permissions like any other group or user, without being associated directly with a person. Their authentication has an expiration date, and so can be useful for temporary projects.
- **SQL Authentication** – Connect to the SQL database with a username and password.

NOTE: If you are using a self-hosted integration runtime to connect to your resource, system-assigned and user-assigned managed identities will not work. You need to use service principal authentication or SQL authentication.

Register and scan SQL Database using Azure Purview

This section will enable you to register the Azure SQL Database data source and set up a scan.

Register the data source

It's required to register the data source in Microsoft Purview before setting up a scan.

1. Open your Microsoft Purview account, and select **Open Microsoft Purview Governance Portal**.

[Refresh](#) [Delete](#)

Resource group : ContosoRG	Type : Microsoft Purview account
Status : Succeeded	Default account : No
Location : West US 2	Platform size : 1 capacity units
Subscription : ContosoSubscription	
Subscription ID : ee2442ec6-7544-4e27-b768-1172cdc01efa	
Tags (edit) : Click here to add tags	

[Get Started](#)

All roles to access Microsoft Purview Governance Portal are assigned by Microsoft Purview account collection admin in Microsoft Purview Governance Portal. [Learn more](#)

Open Microsoft Purview Governance Portal

Start using the unified data governance service and manage your hybrid data estate.

[Open](#)

Manage users

Grant users access to open Microsoft Purview Governance Portal.

[Go to Access control](#)

Documentation

Learn how to be productive quickly. Explore concepts, tutorials, and other guidance available.

[Open](#)

2. Create the collection hierarchy using the **Collections** menu, and assign permissions to individual subcollections, as required.

Collections

adpurview

[+ Add a collection](#)

[Edit role assignments](#)

Collection admins

Name	Type
Contoso Admin contosoadmin@contoso.com	User

Data source admins

Name	Type
Contoso Admin contosoadmin@contoso.com	User

3. Navigate to the appropriate collection under the **Sources** menu, and then select **Register** to register a new SQL Database.
4. Select the Azure SQL Database data source, and then select **Continue**.
5. Provide a name for the data source, select an Azure subscription, select the SQL Database server name, and then select **Apply**.
6. The Azure SQL Database will appear under the selected collection.

Create a scan

To create and set up a scan, follow these steps:

1. Open your Microsoft Purview account and select the **Open Microsoft Purview governance portal**.

Resource group : [ContosoRG](#)
 Status : Succeeded
 Location : West US 2
 Subscription : [ContosoSubscription](#)
 Subscription ID : ee2442ec6-7544-4e27-b768-1172cdc01efa
 Tags (edit) : [Click here to add tags](#)

Type : Microsoft Purview account
 Default account : [No](#)
 Platform size : 1 capacity units

Get Started

All roles to access Microsoft Purview Governance Portal are assigned by Microsoft Purview account collection admin in Microsoft Purview Governance Portal. [Learn more](#)

Open Microsoft Purview Governance Portal
 Start using the unified data governance service and manage your hybrid data estate.
[Open](#)

Manage users
 Grant users access to open Microsoft Purview Governance Portal.
[Go to Access control](#)

Documentation
 Learn how to be productive quickly. Explore concepts, tutorials, and other guidance available.
[Open](#)

2. Select the **Data map** icon, then **Sources** to view the collection hierarchy.
3. Select the **New Scan** icon under the Azure SQL Database registered earlier.
4. Provide a name for the scan, select **Enter manually** for **Database selection method** property, enter the **Database name**, and select the **Credential**. Choose the appropriate collection for the scan, and select **Test connection** to validate the connection. If the connection is successful, select **Continue**.

Scope and run the scan

To scope and run the scan, follow these steps:

1. You can scope your scan to specific database objects by choosing the appropriate items in the list.
2. Select a scan rule set. You can choose between the system default, existing custom rule sets, or create a new rule set inline.
3. Select **New scan rule set**, and provide a new scan rule set name.
4. You can then select the classification rules to be included in the scan rule, and then select **Create**.
5. The **Select a scan rule set** page will the scan rule set you've created.
6. On the **Set a scan trigger** page, configure your scan trigger. Select **Continue**.
7. Review your scan, and then select **Save and run**.

View a scan

To view a scan, follow these steps:

1. Navigate to the data source in the collection, and then select **View Details** to check the status of the scan.
2. The scan details indicate the progress of the scan in the **Last run status** and the number of assets scanned and classified. The **Last run status** will be updated to **In progress** and then **Completed** once the entire scan has run successfully.

Manage Scan

Scans can be managed or run again on completion:

1. Select your scan name to manage the scan.
2. From the scan history page, you can run the scan again, edit the scan, or delete the scan.
3. You can also run an incremental scan or a full scan again.

Data lineage

Generally, data lineage represents the journey the data takes from its origin to where it moves across the data estate over time. Among its many uses are troubleshooting, tracing the root cause in data pipelines, and debugging.

Microsoft Purview Data Catalog connects with other data storage, processing, and analytics platforms to collect lineage information. As a result, the Catalog contains a generic, scenario-specific lineage experience.

Microsoft Purview supports data lineage from Azure SQL Database. At the time of setting up a scan, you can enable lineage extraction toggle button to extract lineage information.

Prerequisites for setting up scan with Lineage extraction

1. Follow steps under authentication for a scan using Managed Identity section to authorize Microsoft Purview scan your Azure SQL Database.
2. Sign in to Azure SQL Database with Azure AD account and assign proper permission (for example: **db_owner**) to Purview Managed identity. Use below example SQL syntax to create user and grant permission by replacing **purview-account** with your account name.

```
CREATE user <purview-account> FROM EXTERNAL PROVIDER  
GO  
EXEC sp_addrolemember 'db_owner', <purview-account>  
GO
```

3. Run below command on your Azure SQL Database to create a master key.

```
CREATE MASTER KEY  
GO
```

Create scan with lineage extraction toggle turned on

1. Enable lineage extraction toggle in the scan screen.

2. Select your method of authentication by following steps in the scan section.
3. Once the scan is successfully set up from previous step, a new scan type called Lineage extraction will run incremental scans every 6 hours to extract lineage from Azure SQL Database. Lineage is extracted based on the actual stored procedure runs in the Azure SQL Database.

Search Azure SQL Database assets and view runtime lineage

You can browse the data catalog or search the data catalog to view asset details for Azure SQL Database following the steps below:

1. Go to asset -> lineage tab, you can see the asset lineage when applicable. Refer to the supported capabilities section on the supported Azure SQL Database lineage scenarios. For more information about lineage in general, see data lineage and lineage user guide
2. Go to stored procedure asset -> Properties -> Related assets to see the latest run details of stored procedures
3. Select the stored procedure hyperlink next to Runs to see Azure SQL Stored Procedure Run Overview. Go to properties tab to see enhanced run time information from stored procedure. For example: executedTime, rowCount, Client Connection, and so on

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Where is the data from data classification stored in SQL Server 2019?

- In the extended properties for each object
- In the sys.sensitivity_classifications catalog view
- In the sys.all_columns catalog view

Check Answers

Multiple choice

Which server-level action group audits queries and stored procedures executed against an Azure SQL Database?

- BATCH_STARTED_GROUP
- BATCH_COMPLETED_GROUP
- SENSITIVE_BATCH_COMPLETED_GROUP

Check Answers

Multiple choice

Which block predicate prevents users from updating rows to values that violate its row-level security predicate?

- AFTER INSERT
- BEFORE UPDATE
- AFTER UPDATE

Check Answers

Multiple choice

Which of the following features can be used to automate data discovery through the provision of data scanning and classification as a service?

- Database auditing
- Dynamic data masking
- Azure Purview

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

This module explored the practices of setting compliance controls on the data that is stored within the SQL Server database. You also reviewed several security features available for Azure SQL, including the Microsoft Defender for SQL.

Now that you've reviewed this module, you should be able to:

- How data should be classified
- Why server and database audit are important
- How to implement row level security and dynamic data masking
- Understand the usage of Microsoft Defender for SQL
- How Azure SQL Database Ledger works
- Explore Azure Purview supported capabilities

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

Introduction

A major part of the job of a database administrator is proper performance monitoring. This task does not change when moving to a cloud platform. While Azure offers tools for monitoring, you may lack some specific controls around hardware that you would have in an on-premises environment which makes understanding how to identify and resolve performance bottlenecks while in Azure SQL that is much more critical.

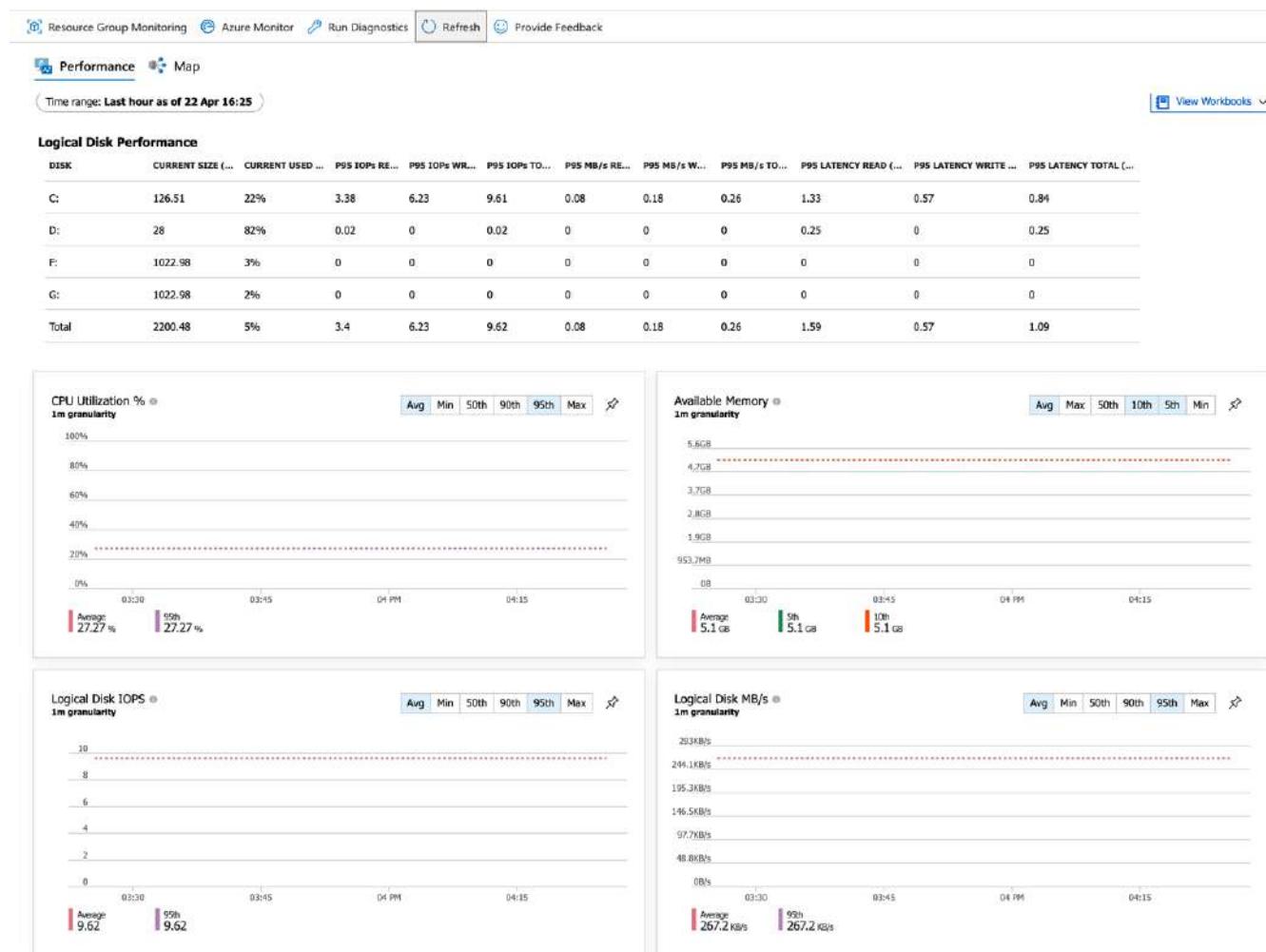
Learning objectives

- Understanding methods to review potential performance issues
- Identify critical Azure metrics
- Learn how to collect metrics for an established baseline
- Use extended events for performance analysis
- Understand Azure SQL Insights

Describe performance monitoring tools

Azure provides multiple methods to monitor the performance of your resources and create a baseline. Each method can be tailored for a specific metric. The metrics that you can monitor will vary depending on the type of Azure resource you are monitoring. For example, Azure SQL Database and SQL Server on an Azure Virtual Machine will have different metrics available in the Azure portal.

The following examples are focused on an Azure Virtual Machine. When you deploy an Azure Virtual Machine from the Azure Marketplace, an agent is installed in the virtual machine that provides a basic set of operating system metrics that are presented to you in the Azure portal. This agent supplies metrics to a service called Azure Monitor, which is a comprehensive platform monitoring solution that collects and displays a standard set of metrics from Azure resources. In the case of a virtual machine, the default metrics captured are CPU, network utilization, and disk read and write operations. You can capture additional metrics beyond what is captured in Azure monitor by enabling Monitoring Insights for your virtual machine as shown in the following image.



These metrics pertain to the operating system, not SQL Server. You'll notice that the namespace for each metric is the virtual machine host, not SQL Server.

You are unable to view SQL Server-specific metrics from within the portal. For detailed SQL Server-specific metrics you will need to gather them from the virtual machine itself.

Azure Monitoring Insights allows you to collect additional data points like storage latency, available memory, and disk capacity. These Azure Monitor Insights can be one way of viewing a baseline of performance for your Azure

Virtual Machine including I/O performance, memory, and CPU utilization. This data is stored in an Azure Log Analytics workspace. Azure Log Analytics is the primary tool in Azure for storing and querying log files of all kinds. Log Analytics is queried by a SQL-like language called Kusto Query Language (KQL).

If you create a virtual machine with one of the pre-configured SQL Server images in the Azure Marketplace, you can also get the SQL virtual machine resource provider as shown in the following image.

Resource group (change) : SQL2019

Status : Online
Location : East US 2
Subscription (change) : Visual Studio Ultimate with MSDN
Subscription ID :
SQL configuration : Click here to view SQL extension configuration
Tags (change) : Click here to add tags

Version	: SQL Server 2019
Edition	: Developer
Virtual machine	: VM1
Virtual machine OS	: Windows (Windows Server 2019 Datacenter)
Virtual machine size	: Standard DS3 v2
License type	: Pay As You Go

Notifications (0) Features (7)

- License type** : CONFIGURED
- Storage configuration** : CONFIGURED
- SQL connectivity** : CONFIGURED
- Azure Key Vault integration** : NOT CONFIGURED
- Automated patching** : CONFIGURED
- Automated backup** : NOT CONFIGURED
- R Services (Advanced analytics)** : NOT CONFIGURED

Disk Usage

Category	Size
SQL data	9 GiB
SQL log	921 MiB
System Reserved	91.2 GiB
Other	2.1 TiB
Total Available	1,200GB

You can launch this screen in the Azure portal by going to the **Settings** section of the main blade for an Azure Virtual Machine, then clicking on the **SQL Server configuration** option. To see the view from the screen above, click on **Manage SQL Virtual Machine**.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

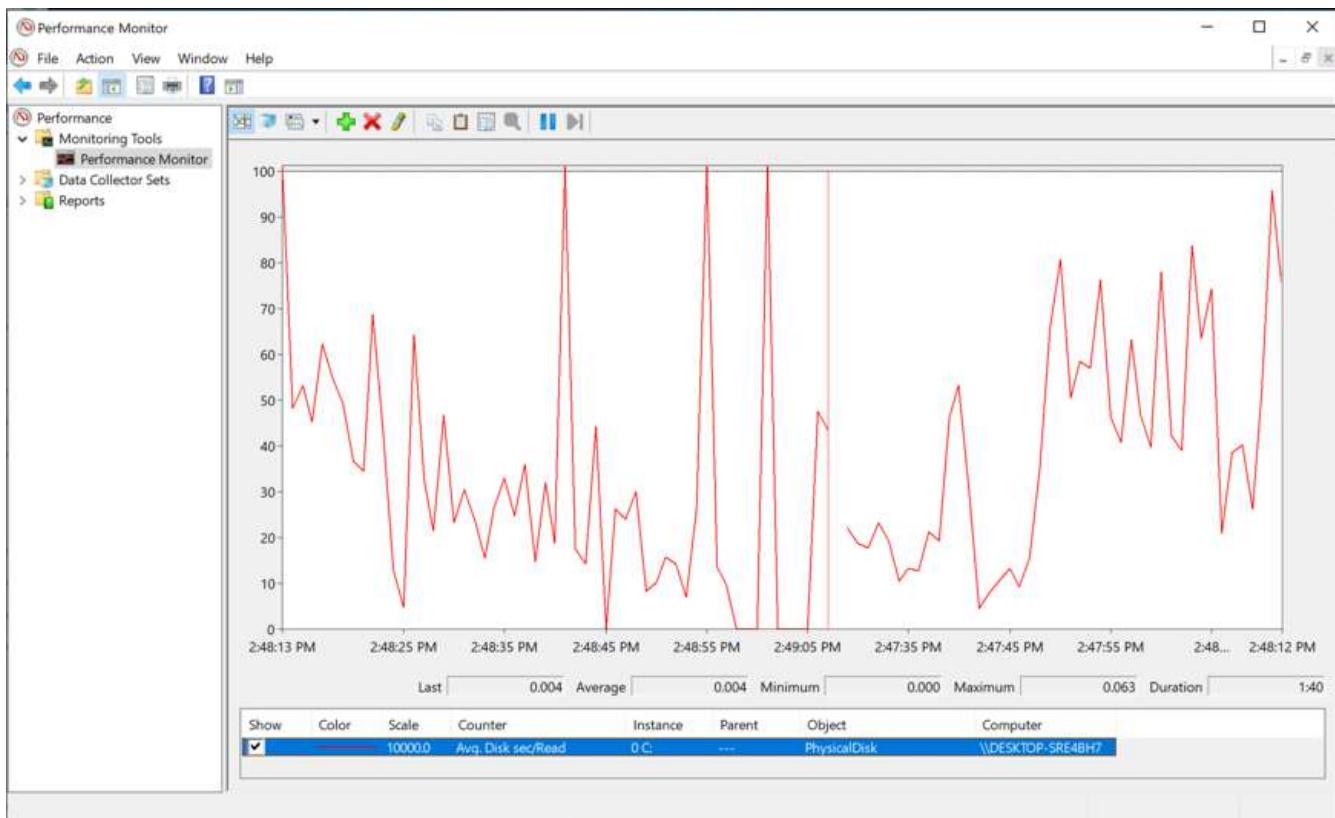
The screenshot shows the Azure portal interface for managing a virtual machine named DP300VM. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Networking, Connect, Disks, Size, Security, Advisor recommendations, Extensions, Continuous delivery, Availability + scaling, Configuration, Identity, SQL Server configuration), Properties, Locks, Operations (Bastion, Auto-shutdown), and a search bar. The 'SQL Server configuration' link is highlighted. The main content area is titled 'SQL management experience on Virtual Machines' and discusses the new SQL focused management experience, mentioning automated patching, backup, and edition flexibility. It also notes that earlier manageability features were limited to specific marketplace images but are now available for any Azure virtual machine with SQL Server installed. A blue button labeled 'Manage SQL virtual machine' is present.

Take note that to access future features and product improvements, you will want to additionally [register your SQL Server VM with the SQL IaaS Agent extension](#).

This dashboard allows you to see how much space your database files and transaction log file are consuming, and allows you to manage the features provided by the resource provider like automated patching and storage configuration. You can manually install the SQL Resource Provider for other installations of SQL Server on Azure Virtual Machine that were not defined as part of the virtual machine.

Performance Monitor with SQL Server on an Azure Virtual Machine

Whether you are using an on-premises server or on an Azure Virtual Machine, the Windows Server platform has a native tool called **Performance Monitor** (commonly shortened to *perfmon* after the name of its executable file) that allows you to easily and routinely monitor performance metrics. *Perfmon* operates with counters for both the operating systems and installed programs. When SQL Server is installed on the operating system, the database engine creates its own group of specific counters.



The above image shows the reporting interface of Performance Monitor, with a single counter being collected. This screen is reached from launching Performance Monitor in Windows and shows a live tracker of a specific performance counter. In many cases, you will capture multiple counters in the same session. *Perfmon* data can be stored locally and analyzed, but in larger environments you can forward performance monitor results into Azure Monitor, where you can have a single view across many servers.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Describe critical performance metrics

You have seen about how to collect data in both Azure Monitor and Windows Performance Monitor. You will now learn how to create metrics in Azure Monitor, which allow you to trigger alerts or execute automated error responses.

Review of Azure metrics

The Azure Monitor service includes the ability to track various metrics about the overall health of a given resource. Metrics are gathered at regular intervals and are the gateway for alerting processes that will help to resolve issues quickly and efficiently. Azure Monitor Metrics is a powerful subsystem that allows you to not only analyze and visualize your performance data, but to also trigger alerts that notify administrators or automated actions that can trigger an Azure Automation runbook or a webhook. You also have the option to archive your Azure Metrics data to Azure Storage, since active data is only stored for 93 days.

Create metric alerts

Utilizing the Azure portal, you can create alert rules, based on defined metrics, in the overview section of the Azure Monitor blade. Azure Monitor Alerts can be scoped in three ways. For example, using Azure Virtual Machines as an example you can specify the scope as:

- A list of virtual machines in one Azure region within a subscription
- All virtual machines (in one Azure region) in one or more resource groups in a subscription
- All virtual machines (in one Azure region) in one subscription

In this manner, you can create an alert rule based on resources contained within resource groups as shown.

The screenshot shows the Microsoft Azure Monitor | Alerts interface. On the left, there's a navigation menu with various service links like Home, Dashboard, All services, etc., and a 'Monitor' link which is highlighted with a red box. Below it is a detailed sidebar for 'Monitor | Alerts' with sections for Overview, Activity log, Metrics, Logs, Service Health, Workbooks, Insights (Applications, Virtual Machines, Storage Accounts, Containers, Networks, Cosmos DB), Settings (Diagnostics settings, Autoscale), and Support + Troubleshooting (Usage and estimated costs). At the top right, there are buttons for 'New alert rule', 'Manage alert rules', 'Manage actions', 'View classic alerts', and 'Refresh'. A message at the top says 'Don't see a subscription? Open Directory + Subscription settings'. The main area displays a pie chart and a bar chart with the text 'All is good! You have 33 selected'.

The example shown below reflects a virtual machine named SQL2019 on which we are creating an alert that is at the scope of the individual virtual machine.

This screenshot shows the Azure Virtual machines blade for a VM named 'SQL2019'. The 'Alerts' section in the sidebar is highlighted with a red box. The main content area shows the 'SQL2019 | Alerts' blade with a similar layout to the previous one, featuring a sidebar with Operations (Bastion, Auto-shutdown, Backup, Disaster recovery, Update management, Inventory, Change tracking, Configuration management, Policies, Run command) and Monitoring (Insights, Metrics, Diagnostic settings) sections, and a top navigation bar with 'New alert rule', 'Manage alert rules', 'Manage actions', and 'Subscription' dropdown set to 'Dev-Test-Lab'.

Regardless the scope of the alert, the creation process is the same.

From the alerts screen, click on New Alert Rule. If an alert is created from within the scope of a resource, the resource values should be populated for you. You can see that the resource is the SQL2019 virtual machine, the subscription is Dev-Test-Lab and the resource group in which it resides is SQLPlayground.

Under the Condition section, click **Add**:

Dashboard > Virtual machines > SQL2019 | Alerts > Create rule

Create rule

Rules management

*** RESOURCE**

SQL2019

Select

HIERARCHY

Dev-Test-Lab > SQLPlayground

*** CONDITION**

No condition defined, click on 'Add' to select a signal and define its logic

Add

ACTIONS GROUPS (optional)

Action group name

No action group selected

Contain actions

Add Create

Action rules (preview) allows you to define actions at scale as well as suppress actions. Learn more about this functionality by clicking on this banner. [this banner](#).

ALERT DETAILS

Alert rule name * [\(i\)](#)

Specify alert rule name. Sample: 'Percentage CPU greater than 70'

Description

Specify alert description here...

This document belongs to: *srinivas*, *sg.dba@outlook.com*, *nauthorised copies allowed!*, *Richard Jillella.*

Select the metric that you wish to alert on. The following image shows Percentage CPU, which you will then see selected.

Configure signal logic

Choose a signal below and configure the logic on the next screen to define the alert condition.

Signal type ⓘ

All

Monitor service ⓘ

All

Displaying 1 - 20 signals out of total 50 signals

Search by signal name

Signal name	↑↓	Signal type	↑↓	Monitor service	↑↓
Percentage CPU	↗↘	Metric		Platform	
Network In Billable (Deprecated)	↗↘	Metric		Platform	
Network Out Billable (Deprecated)	↗↘	Metric		Platform	
Disk Read Bytes	↗↘	Metric		Platform	
Disk Write Bytes	↗↘	Metric		Platform	
Disk Read Operations/Sec	↗↘	Metric		Platform	
Disk Write Operations/Sec	↗↘	Metric		Platform	

The alerts can be configured in a static manner (for example, raise an alert when CPU goes over 95%) or in a dynamic fashion using Dynamic Thresholds. Dynamic Thresholds learn the historical behavior of the metric and raise an alert when the resources are operating in an abnormal manner. These Dynamic Thresholds can detect seasonality in your workloads and adjust the alerting accordingly.

If Static alerts are used, you must provide a threshold for the selected metric. In this example, 80 percent was specified. This threshold means that if the CPU utilization exceeds 80 percentage over a given period, an alert will be fired and react as specified.

Both types of alerts offer Boolean operators such as the 'greater than' or 'less than' operators. Along with Boolean operators, there are aggregate measurements to select from such as average, minimum, maximum, count, average, and total. With these options available, it's easy to construct a flexible alert that will suit just about any enterprise level alerting.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!



Configure signal logic



Alert logic

Threshold ⓘ

Static

Dynamic

Operator ⓘ

Greater than

Aggregation type * ⓘ

Average

Threshold value * ⓘ

80

✓

%

Condition preview

Whenever the percentage cpu is greaterthan 80 %

Evaluated based on

Aggregation granularity (Period) * ⓘ

5 minutes

Frequency of evaluation ⓘ

Every 1 Minute

Done

After creating the alert, in order to notify administrators or launch an automation process, an action group needs to be configured.

NOTE: Defining an action group is optional, and if one is not configured the alert will just log the notification to storage with no further action is taken. You can create a new action group from the metrics screen, by clicking **Add** next to Action Groups. You will then see this dialog:

Select an action group to attach to this alert rule

The action group selected will attach to this alert rule

+ Create action group

Subscription Create action group

Contoso Ltd



Once you click Create Action group, you will see the screen shown below. You will name the action group and define an alert and the response. In this example, the administrator is going to be emailed in the event of the alert's

condition being triggered.

You can configure the email or SMS details as can be seen below. You can reach this screen either by clicking **Edit Details** under **Configure**, or by adding a new action, which will also bring up the configuration screen.

With an action group, there are several ways in which you can respond to the alert. The following options are available for defining the action to take:

- Automation Runbook
- Azure Function
- Email Azure Resource Manager Role
- Email/SMS/Push/Voice
- ITSM
- Azure Logic App
- Secure Webhook
- Webhook

There are two categories to these actions—notification, which means notifying an administrator or group of administrators of an event, and automation, which is taking a defined action to respond to a performance condition.

Review older performance data

One of the benefits of utilizing the Azure Monitor is the ability to easily and quickly review past metrics that were gathered. If you examine a resource, you'll note a datetime picker in the upper right-hand corner. Azure Monitor Metrics will be retained for 93 days, after which they are purged, however you do have the option to archive them to Azure Storage.

You are also able to select a smaller window of time such as the last 30 minutes, last hour, last 4 hours, or last 12 hours as an example. The flexibility of Azure monitor allows administrators to quickly identify issues as well as to potentially diagnose past issues.

SQL Server metrics that matter

Microsoft SQL Server is a well instrumented piece of software that collects a great deal of performance metadata. The database engine has metrics that can be monitored to help identify and improve performance-related issues. Some operating system metrics are only viewable from within performance monitor while others can be accessed through T-SQL queries, in particular, by selecting from the dynamic management views (DMVs). There are some metrics that are exposed in both locations so knowing where to identify specific metrics is important. One example of data that can only be captured from DMVs is data and transaction log file read/write latency as exposed in `sys.dm_os_volume_stats`. On the other hand, an example of an OS metric that is not available directly through SQL Server is the seconds per disk read and write for the disk volume. Combining these two metrics can

help you gain better understand if a performance issue is related to database structure or a physical storage bottleneck.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Describe critical performance metrics

A baseline is a collection of data measurements that helps you understand the normal “steady state” of your application or server’s performance. Having the data collected over time allows you to identify changes from the normal state. Baselines can be as simple as a chart of CPU utilization over time, or complex aggregations of metrics to offer granular level performance data from specific application calls. The granularity of your baseline will depend on the criticality of performance of your database and application.

With any type of application workload, it is imperative to establish a working baseline. A baseline will help you identify if an ongoing issue should be considered within normal parameters or has exceeded given thresholds. Without a baseline, every issue encountered could be considered normal and therefore not require any additional intervention.

Correlating SQL Server and operating system performance

When deploying SQL Server on an Azure virtual machine, it’s critical to correlate the performance of SQL Server with the performance of the underlying operating system. If you are using Linux as the operating system, you will need to install InfluxDB, Collectd, and Grafana to capture data similar to Windows Performance Monitor. These services collect data from SQL Server and provide a graphical interface to review the data. Utilizing these tools on Linux or Performance Monitor on Windows can be used in conjunction looking at SQL Server-specific data such as SQL Server wait statistics. Using these tools together will allow you to identify bottlenecks in hardware or code. The following Performance Monitor counters are a sampling of useful Windows metrics, and can allow you to capture a good baseline for a SQL Server workload:

Processor(_Total)% Processor Time - This counter measures the CPU utilization of all of the processors on the server. It is a good indication of the overall workload, and when used in conjunction with other counters, this counter can identify problems with query performance.

Paging File(_Total)% Usage - In a properly configured SQL Server, memory should not page to the paging file on disk. However, in some configurations you may have other services running that consume system memory and lead to the operating system paging memory to disk resulting in performance degradation.

PhysicalDisk(_Total)\Avg. Disk sec/Read and Avg. Disk sec/Write - This counter provides a good metric for how the storage subsystem is working. Your latency values in most cases should not be above 20 ms, and with Premium Storage you should see values less than 10 ms.

System\Processor Queue Length - This number indicates the number of threads that are waiting for the time on the processor. If it is greater than zero, it indicates CPU pressure, indicating your workload could benefit from more CPUs.

SQLServer:Buffer Manager\Page life expectancy - Page life expectancy indicates how long SQL Server expects a page to live in memory. There is no proper value for this setting. Older documentation refers to 300 seconds as proper, but that was written in a 32-bit era when servers had far less RAM. You should monitor this value over time, and evaluate sudden drops. Such drops in the counter’s value could indicate poor query patterns, external memory pressure (for example, the server running a large SSIS package) or could just be normal system processing like running a consistency check on a large database.

SQLServer:SQL Statistics\Batch Requests/sec - This counter is good for evaluating how consistently busy a SQL Server is over time. Once again there is no good or bad value, but you can use this counter in conjunction with % Processor time to better understand your workload and baselines.

SQLServer:SQL Statistics\SQL Compilations/sec and SQL Re-Compilations/sec - These counters will be updated when SQL Server has to compile or recompile an execution plan for a query because there is no existing plan in the plan cache, or because a plan was invalidated because of a change. Recompiles can indicate T-SQL with recompile query hints, or be indicative of memory pressure on the plan cache caused by either many ad-hoc queries or simple memory pressure.

These counters are just a sample of the available performance monitor counters that are available to you. While the above counters provide a good baseline of performance, you may need to examine more counters to identify specific performance problems.

Wait statistics

When a thread is being executed and is forced to wait on an unavailable resource, SQL Server keeps track of these metrics. This information is easily identifiable via the dynamic management view (DMV)

`sys.dm_os_wait_stats`. This information is important to understanding the baseline performance of your database, and can help you identify specific performance issues both with query execution and hardware limitations. Identifying the appropriate wait type and corresponding resolution will be critical for resolving performance issues. Wait statistics are available across the Azure SQL platform.

Explore extended events

The extended events engine in Azure SQL is a lightweight and powerful monitoring system that allows you to capture granular information about activity in your databases and servers. The monitoring solutions on the Azure platform allow you to easily configure powerful monitoring for your environment and provide automated responses to error conditions.

Extended events build on the functionality of SQL Server Profiler by allowing you to trace queries and by exposing additional data (events) that you can monitor. Some examples of issues you might troubleshoot with Extended Events include:

- Troubleshooting blocking and deadlocking performance issues.
- Identifying long-running queries.
- Monitoring Data Definition Language (DDL) operations.
- Logging missing column statistics.
- Observing Memory Pressure in your database.
- Long-running physical I/O operations.

The extended event framework also allows you to use filters to limit the amount of data you collect in order to reduce the overhead of data collection, and allows you to more easily identify your performance problem by targeting your focus onto specific areas.

Below is an example of an extended event session created on Azure SQL Database:

The screenshot shows the SSMS Object Explorer and a results grid. The Object Explorer tree on the left shows the database structure, with a red box highlighting the 'Extended Events' node under 'AdventureWorks'. Inside 'Extended Events', there is a 'Sessions' node, which contains a 'xe_deadlocks' node, and under 'xe_deadlocks', there is a 'package0.event_counter' node. The right pane displays a table titled 'AdventureWorks - x...cks: event_counter' with the following data:

package_name	event_name	count
sqlserver	lock_deadlock	1
sqlserver	database_xml_deadlock_report	1
sqlserver	lock_deadlock_chain	2

In the image above, `xe_deadlocks` is the name of an extended event session running on `AdventureWorks` database (on the left side of the image). The `event_counter` target node, which is under your event session node, counts the number of occurrences of each event in the event session. To view the target data in the SSMS Object Explorer,

you can right-click the target node, and then select *View Target Data*. SSMS displays the data as we see on the left side of the image, and the count results for each event.

For more information about extended events on Azure SQL Database, see [Extended events in Azure SQL Database](#).

What can I monitor with extended events?

Extended events cover the full surface area of SQL Server, and are divided into four channels, which define the audience of an event.

- **Admin** - Admin events are targeted for end users and administrators. The events included indicate a problem within a well-defined set of actions an administrator can take. An example of this is the generation of an XML deadlock report to help identify the root cause of the deadlock.
- **Operational** - Operational events are used for analysis and diagnostics or common problems. These events can be used to trigger an action or task based on an occurrence of the event. An example of an operational event would be a database in an availability group changing state, which would indicate a failover.
- **Analytic** - Analytic events are typically related to performance events and are published in high volume. Tracing stored procedure or query execution would be an example of an analytic event.
- **Debug** - Debug events are not necessarily fully documented and you should only use them when troubleshooting in conjunction with Microsoft support.

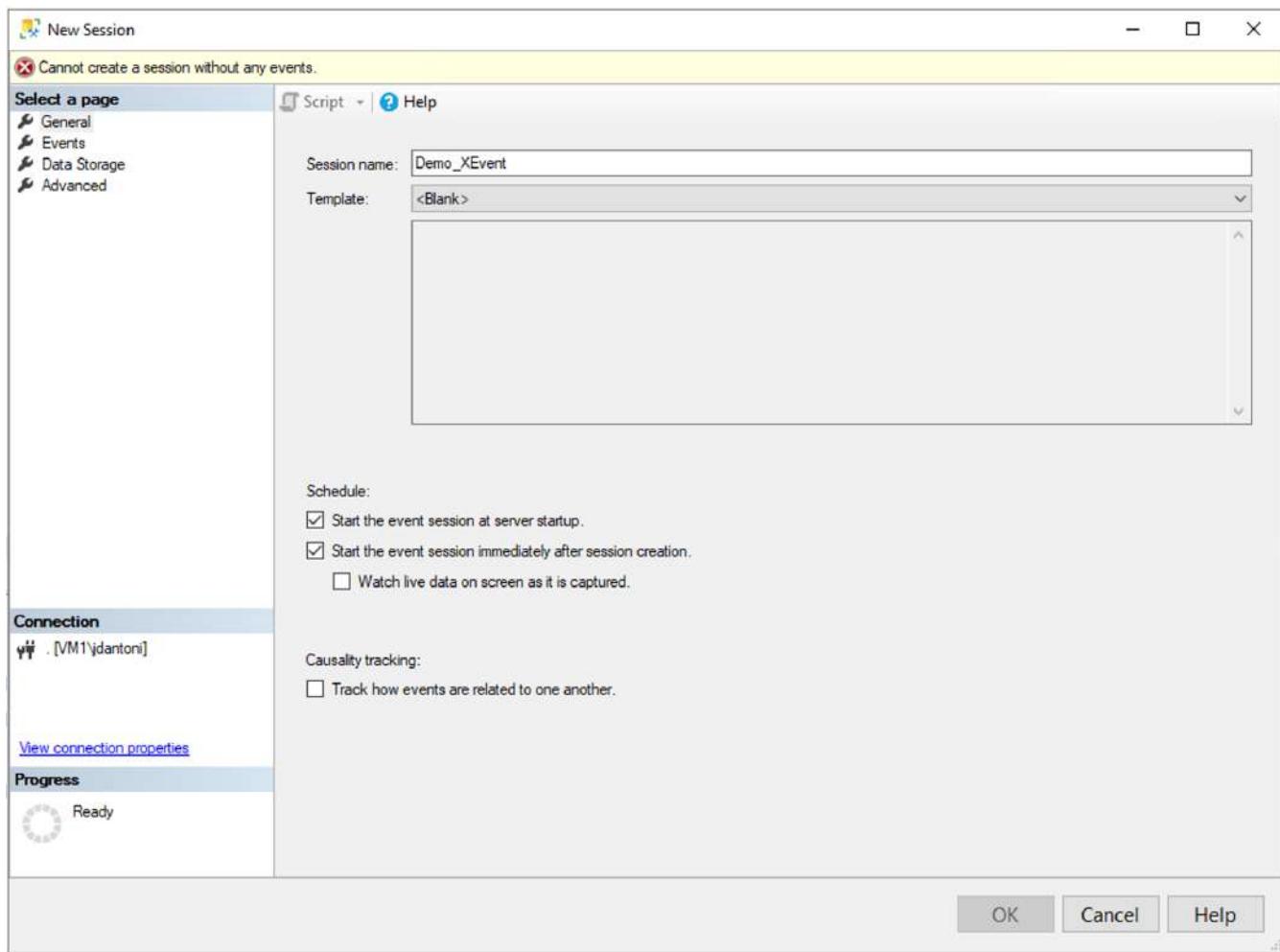
Events are added to sessions, which can host multiple events. Typically, multiple events are grouped together in a session to capture a related set of information.

You can run the query below to obtain a list of the available events, actions, and targets:

```
SELECT
    obj.object_type,
    pkg.name AS [package_name],
    obj.name AS [object_name],
    obj.description AS [description]
FROM sys.dm_xe_objects AS obj
    INNER JOIN sys.dm_xe_packages AS pkg ON pkg.guid = obj.package_guid
WHERE obj.object_type IN ('action', 'event', 'target')
ORDER BY obj.object_type,
    pkg.name,
    obj.name;
```

Create extended events session

You will see below the basic process of creating an extended events session using the *New Session* dialog from SQL Server Management Studio. You can get to this screen by expanding the *Management* node in SSMS, expanding the Extended Events node, right-clicking on Sessions, and selecting *New Session*.

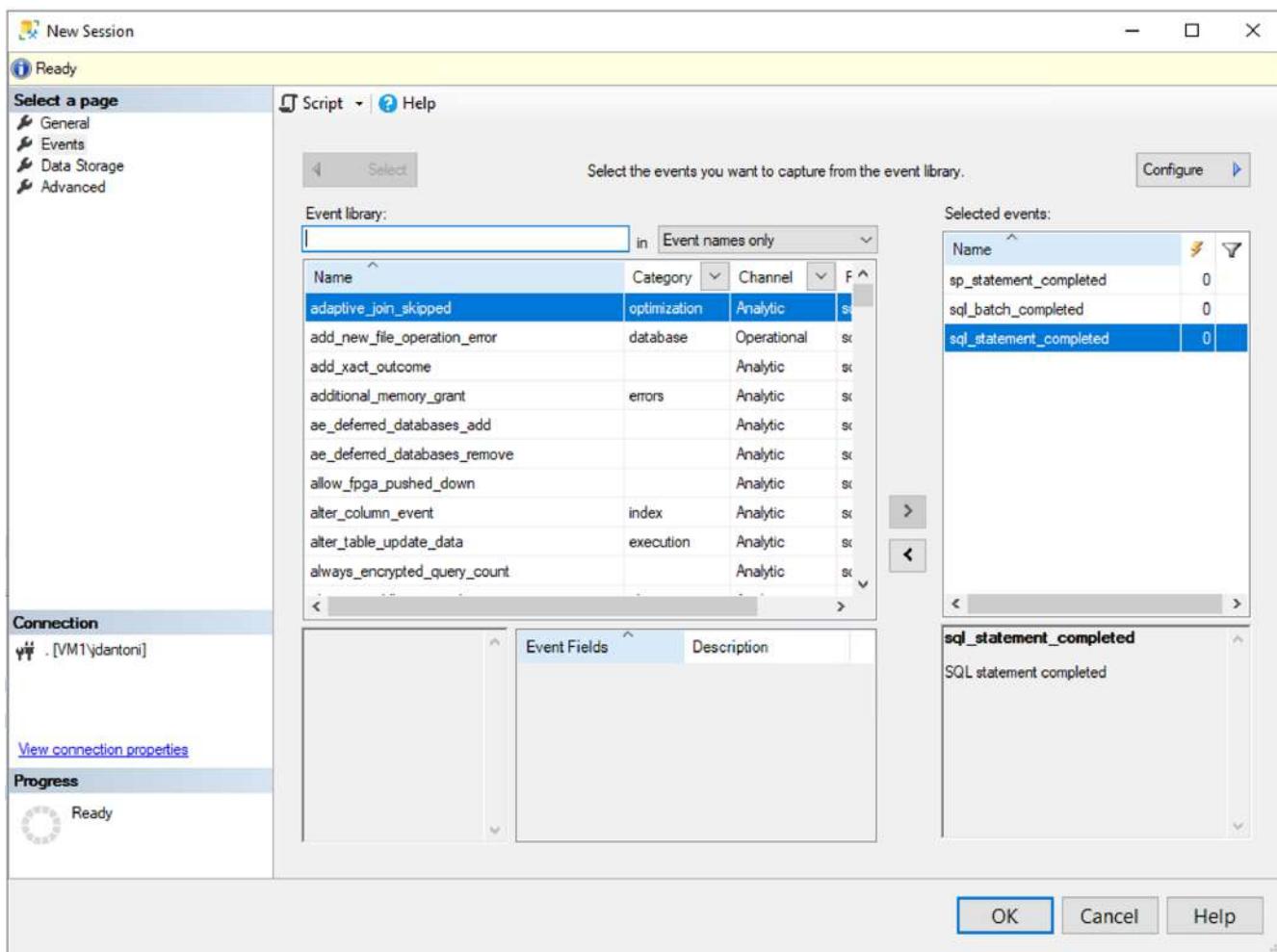


The image above shows the *New Session* dialog for the extended events feature. You must first name the session. SQL Server provides numerous templates which are grouped into the following categories:

- Locks and Blocks
- Profiler Equivalents
- Query Execution
- System Monitoring

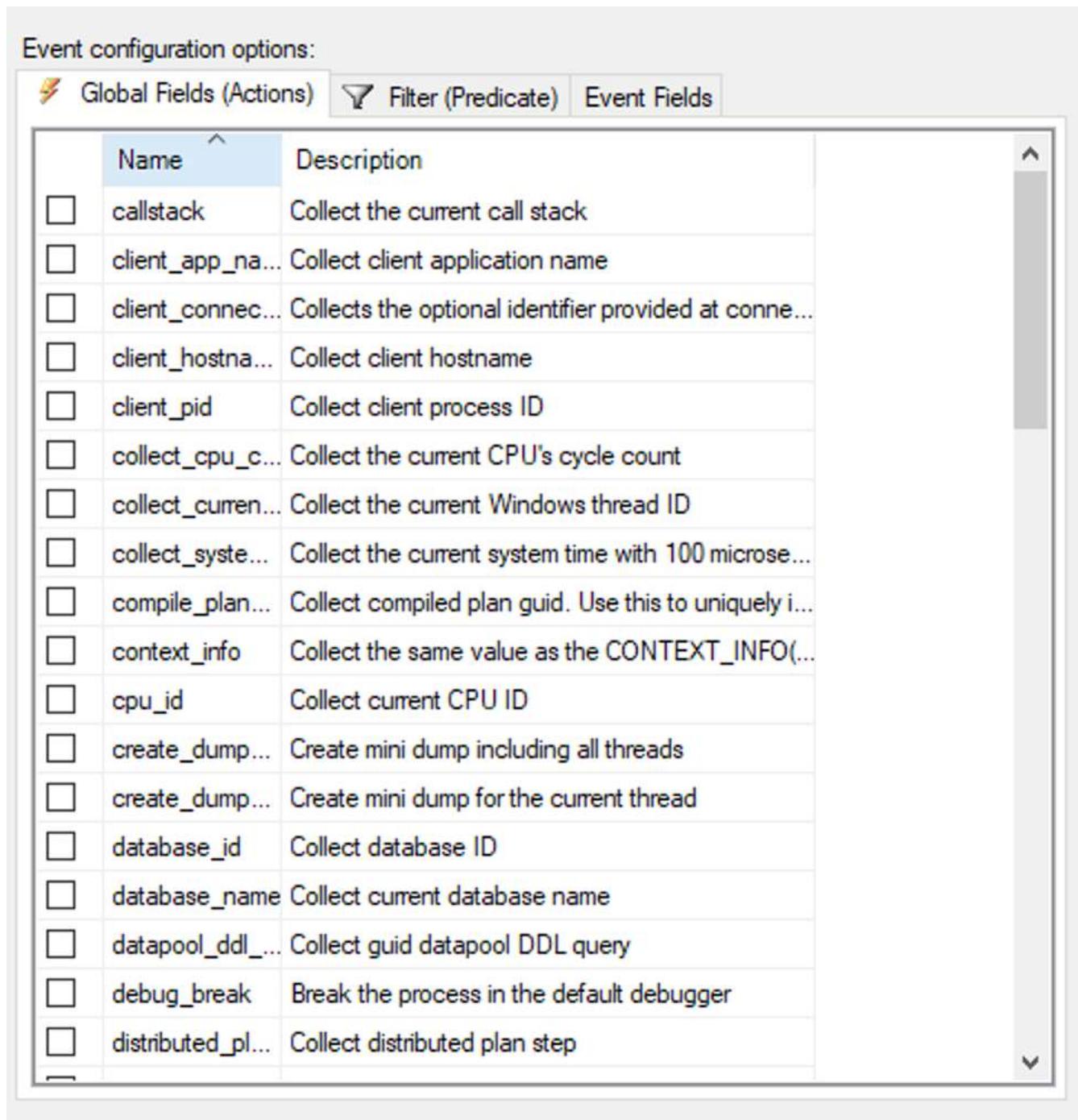
These predefined templates allow you to quickly get started with using extended events for monitoring. In this example, you will see events manually added to the session to walk you through all of the options, but when you are getting started, using a template can be an easy way to create a basic session.

You have a couple of check box options for when to start this session. You can choose to have your new session started whenever the server starts, and you can also choose to start the session as soon as it's been created. Administrators can start, and stop extended event sessions at any time through the *Extended Events* node in SQL Server Management Studio. You also have the option of enabling causality tracking, which adds a globally unique identifier (GUID) and sequence number to the output of each event, which allows you to easily step through the order that the events occurred.



The image above shows the screen where you add the events to your session. An event represents a point of interest within the database engine code — these can represent purely internal system operations, or they can be associated with user actions like query execution. In the above example, you can see that the events `sp_statement_completed`, `sql_batch_completed`, and `sql_statement_completed` have been added to this event session. By default, this session would capture all instances of these events taking place on your instance. You can limit collection by clicking the configure button.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*



The event configuration screen allows for defining what data you are collecting as it relates to your events. Global fields, allow you to choose the data you are collecting, when your event occurs. Global fields are also known as actions, as the action is to add additional data fields to the event. These fields represent the data that is collected when the extended event occurs, and are common across most extended events. The image below shows the filter options for an extended event.

Selected events:

Name	Count	Action
sp_statement_completed	0	✓
sql_batch_completed	0	✓
sql_statement_completed	0	✓

Event configuration options:

Global Fields (Actions) Filter (Predicate) Event Fields

	And/Or	Field	Operator	Value
		sqlserver.is_system	=	0

Click here to add a clause

sql_statement_completed
SQL statement completed

sqlserver.is_system (package0.boolean)
Get whether current session is system

Filters are a powerful feature of Extended Events that allow you to use granular control to capture only the specific occurrences of the event you want to capture. In this example, you can see that filter is being applied on the field `sqlserver.is_system` where it is equal to zero, which indicates that the query is not an internal operation. In other words, the session will not capture completion of statements that are submitted by system connections, and we only want to capture statements submitted by users or user applications.

Filters apply to a single field on a single event. If you want to make sure that you aren't tracing system activities for any events, you'll need a separate filter for each: for the `sql_statement_completed` event, for the `sql_batch_completed` event, and for the `sp_statement_completed` event.

It is good practice to configure a filter for each event that you are capturing. This helps improve the efficiency of data collection, and allows you to narrow the focus of your search.

The image below shows the event fields that are collected. These are specific to the event being triggered, and can include optional fields for collection. In the above event, you can see the optional collection options are `statement`, and `parameterized_plan_handle`.

Once you have defined an event session, you will define a storage target, as shown in the image below.

An extended event session has a target — a target can be simply thought of as a place for the engine to keep track of occurrences of an event. Two of the more common targets are *event file* which is a file on the file system that can store events, and in Azure SQL PaaS offerings this data is written to a blob storage. Another common target is the *ring buffer* which is within SQL Server's memory. The *ring buffer* is most commonly used for live observation of an

event session as it's a circular buffer, and data is not persisted beyond a session. Most targets process data asynchronously, which means that the event data is written to memory before being persisted to disk. The exception is the Event Tracing for Windows target (ETW), and Event Counter targets, which are processed synchronously.

The following table contains information, and uses for each type of Extended Events target.

Target	Description	Processing
Event Counter	Counts all events that occurred during an Extended Event session. This is used to obtain information about workload characteristics about a workload without the overhead of a full event collection.	Synchronous
Event File	Writes event session output from memory onto persistent file on disk.	Asynchronous
Event Pairing	Many events that generally occur in pairs (e.g. lock acquire, lock release), and this collection can be used to identify when those events do not occur in a matched set.	Asynchronous
Event Tracing for Windows (ETW)	Used to correlate SQL Server events with the Windows OS event data.	Synchronous
Histogram	This is similar to event counter, which counts the occurrences of an event. The difference is that the histogram can count based on a specific event column or action.	Asynchronous
Ring Buffer	Used to hold data in memory. Data is not persisted to disk and maybe frequently flushed from the buffer	Asynchronous

Alternatively, you can create an extended events session using T-SQL. The following T-SQL commands provide an example on how to create an extended events session:

```

IF EXISTS (SELECT * FROM sys.server_event_sessions WHERE name='test_session')
    DROP EVENT session test_session ON SERVER;
GO

CREATE EVENT SESSION test_session
ON SERVER
    ADD EVENT sqlos.async_io_requested,
        ADD EVENT sqlserver.lock_acquired
        ADD TARGET package0.etw_classic_sync_target (SET default_etw_session_logfile_path =
N'C:\demo\traces\sqletw.etl')

```

```
WITH (MAX_MEMORY=4MB, MAX_EVENT_SIZE=4MB);
```

```
GO
```

Event sessions can be scoped to a server or a database. In the example shown above, you are adding two events, and using the Event Tracing for Windows (ETW) path with a file location. After you create the session, you'll have to start it. You can do this through T-SQL, and **ALTER** the session using the **STATE** option, or you can use SQL Server Management Studio for it.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Describe Azure SQL Insights

One of the benefits of using any of the products part of the Azure SQL family is the monitoring capability that is built into Azure platform. Beyond the simple Azure Monitor data collection, SQL Insights is a component that allows you to analyze your queries, and tune performance.

With SQL Insights' interactive features, you can customize telemetry collection and frequency, and combine data from multiple sources into a single monitoring experience.

How SQL Insights works

SQL Insights collects data remotely from dynamic management views, and it's built on top of the Azure Monitor platform, giving customers access to the native alerting and out-of-the-box visualizations. It also retains a set of metrics over time, which allows you to investigate performance issues that you may have encountered in the past.

To get started with SQL Insights, you need a dedicated virtual machine that will monitor and remotely collect data from your SQL servers. This dedicated virtual machine needs to have the following components installed:

- Azure Monitor agent
- Workload Insights extension

In order to increase control over charges, customers can also choose which telemetry data to collect, the frequency, and manage retention policy parameters. Database activity and the settings that you've set in your monitoring profiles will determine the amount of data being collected, and the exact cost.

Lastly, you can access performance data from the SQL Insights workbook template, or directly from the monitoring logs.

SQL Insights in Azure Monitor

To get started with SQL Insights, from the **Monitor** blade, select **SQL (preview)**, and then **Create new profile**.

The screenshot shows the Azure Monitor | SQL (preview) interface. On the left, there's a navigation sidebar with links like Overview, Activity log, Alerts, Metrics, Logs, Service Health, Workbooks, Insights (Applications, Virtual Machines, Storage accounts, Containers, Networks), and SQL (preview). The SQL (preview) link is highlighted with a red box. The main area has sections for Subscription (Contoso Subscription), Monitoring profile (with a dropdown and a 'Create new profile' button highlighted with a red box), and Time range (Last 4 hours). Below these are sections for Onboarding to Azure Monitor SQL Insights (Preview), pricing information, and a 'Create new profile' button.

From the **Create new profile** page, configure the following components:

- **Monitoring profile** – group servers, instances or databases to monitor.
- **Log Analytics workspace** – where to send the SQL monitoring data to.
- **Collection settings** – you can customize the data collection for your profile. The default settings cover the majority of monitoring scenarios and usually don't need to be changed.

After you're done, select **Create monitoring profile**.

Create new profile

X

Azure Monitor



Auto refresh: Off

Enable SQL monitoring

1. Create monitoring profile

A profile allows you to group servers, instances or databases to monitor and analyze as a combined set. It allows you to set the scope of monitoring - whether it is collection environments (development or production), application (billing or customer), the collection settings (e.g. high fidelity data collection vs. low), etc.

Subscription

Contoso Subscription



Resource group

myresourcegroup



Profile name

SQLservers-profile



Profile will be created in the same location where the Log Analytics workspace is.

2. Set destination

Specify the Log Analytics workspace to send the SQL monitoring data to.

Workspace subscription

Contoso Subscription



Log Analytics workspace

DefaultWorkspace-xxxxxx-xxxx-xxxx-xxxx-xxxxcdc01efa-EUS



3. Collection settings

Configure SQL monitoring data collection for your profile.

Common settings Customized collection (advanced)

Use this section to configure common collection settings. The default settings cover the majority of monitoring scenarios and usually does not need to be changed.

Collection interval

60 seconds



Azure SQL Database settings

7 selected



Azure SQL Managed Instance settings

7 selected



SQL Server settings

7 selected



4. Create monitoring profile

Use the button below to create your monitoring profile

[Create monitoring profile](#)

Back to the **SQL (preview)** page, select the **Manage profile** tab, and then **Add monitoring machine**.

Home > Monitor

The screenshot shows the Azure Monitor interface for SQL (preview). The top navigation bar includes 'Workbooks', 'Customize', and 'Alerts (0)'. The 'Monitoring profile' dropdown is set to 'MyProfile'. The 'Time range' is 'Last 4 hours'. Below the navigation bar, there are tabs for 'Overview' and 'Manage profile', with 'Manage profile' being the active tab. A red box highlights the 'Add monitoring machine' button. The status bar at the bottom indicates 'Profile: MyProfile | Workspace: DefaultWorkspace-xxxxxx-xxxx-xxxx-xxxxcdc01efa-WUS2'.

NOTE: Make sure you have a dedicated virtual machine created before proceed to the next step. At this time, the only supported virtual machine OS is Ubuntu 18.04.

On the **Add monitoring virtual machine** page, ensure you select the virtual machine name, the SQL Server connection strings, and that the following pre-requisites are satisfied:

- Set permissions for SQL accounts
- Create firewall and networking rules for SQL resource or virtual machine

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Add monitoring virtual machine

Azure Monitor



Select virtual machine

Select a virtual machine you would like for monitoring.

Subscription

Contoso Subscription

Virtual machine

sqlinsightsVM

Add SQL Server connection strings

Add the monitoring config for this virtual machine. It involves providing a set of connection strings for the SQL databases you would like to monitor.

Note that you have the option to define parameters to reference secrets from an Azure Key Vault or for just reuse of tokens across connection strings. [Learn more](#)

 Please configure using the template below.

Important

For a secure configuration, it is required to store the password as a Key Vault secret. [Grant access](#) to the managed identity of monitoring VM and [enable network connectivity](#) from the monitoring VM to the key vault where the secret is stored.

Connection strings

```
{
  "version": 1,
  "secrets": {
    "telegrafPassword": {
      "keyvault": "https://mykeyvault.vault.azure.net/",
      "name": "sqlPassword"
    }
  }
}
```

To learn more about how to enable SQL Insights, see [Enable SQL insights \(preview\)](#).

Limitations

SQL insights has no support or has limited support for the following components:

- Non-Azure instances
- Azure SQL Database elastic pools
- Azure SQL Database running on Basic, S0, S1, and S2 service tiers
- Azure SQL Database serverless tier
- Multiple secondary replicas

- Authentication with Azure Active Directory. Only SQL authentication is supported

Azure SQL Insights is a cloud monitoring solution that brings together performance metrics at scale in a single view.

In addition to visualization and data collection, it has built-in intelligence for troubleshooting activities. Furthermore, it allows for custom monitoring alerts and rules that allow for quick identification and resolution of issues.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

Explore Query Performance Insight

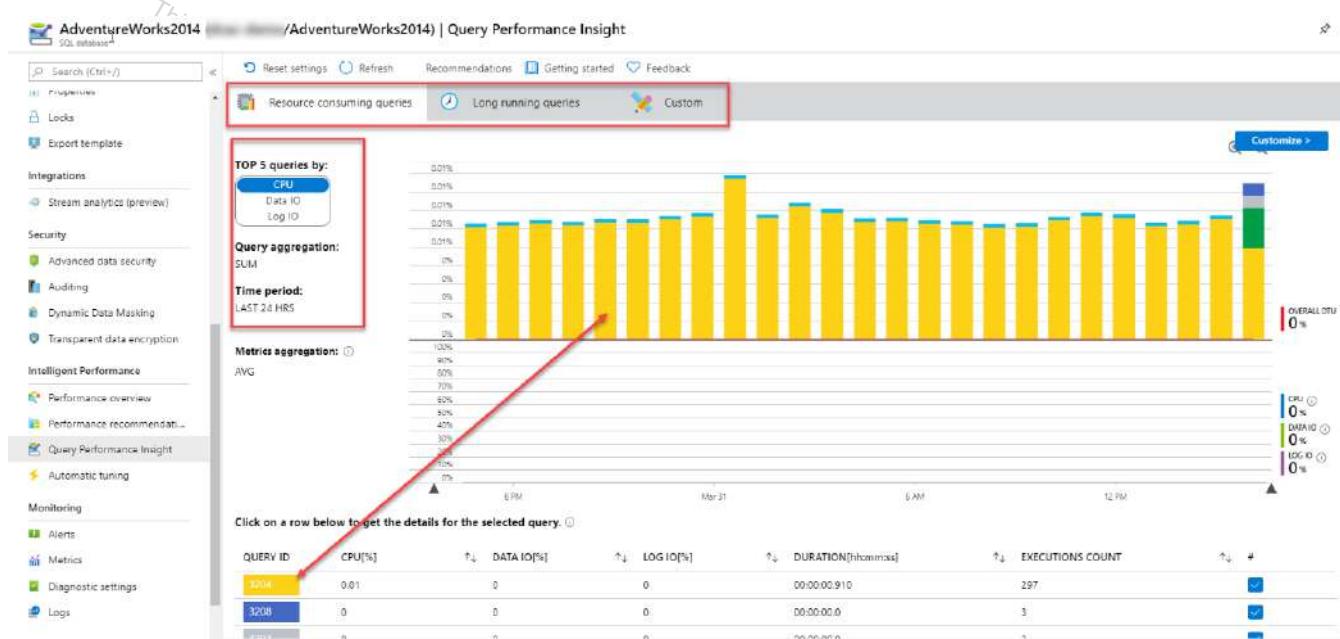
which queries are consuming the most resources is the first step in any database performance tuning endeavor. In older versions of SQL Server, this required extensive tracing and a series of complex SQL scripts, which could make the process of data gathering cumbersome.

Identify problematic queries

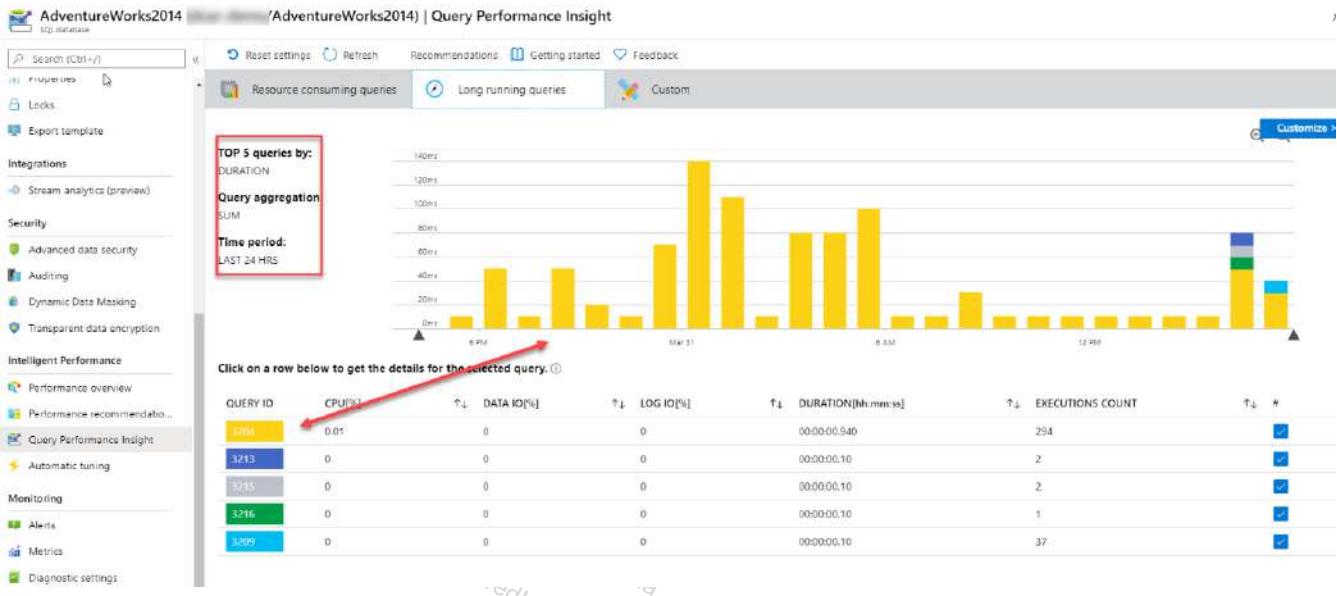
Azure SQL Database offers a tool called Query Performance Insight, that allows the administrator to quickly identify expensive queries. You can navigate to Query Performance Insight in the main blade for your Azure SQL Database in the Intelligent Performance section.

When you launch Query Performance Insight, you'll discover three buttons to allow you to filter for long running queries, top resource consuming queries, or a custom filter. The default value is Resource Consuming Queries. This tab will show you the top five queries sorted by the particular resource that you select on the left. In this case, it was sorted by CPU. You also have other options of sorting by Data IO and Log IO metrics.

You can drill into individual queries by clicking on the row within the lower grid. Each row will be identified with a unique color that correlates to the color within the bar graph above it.

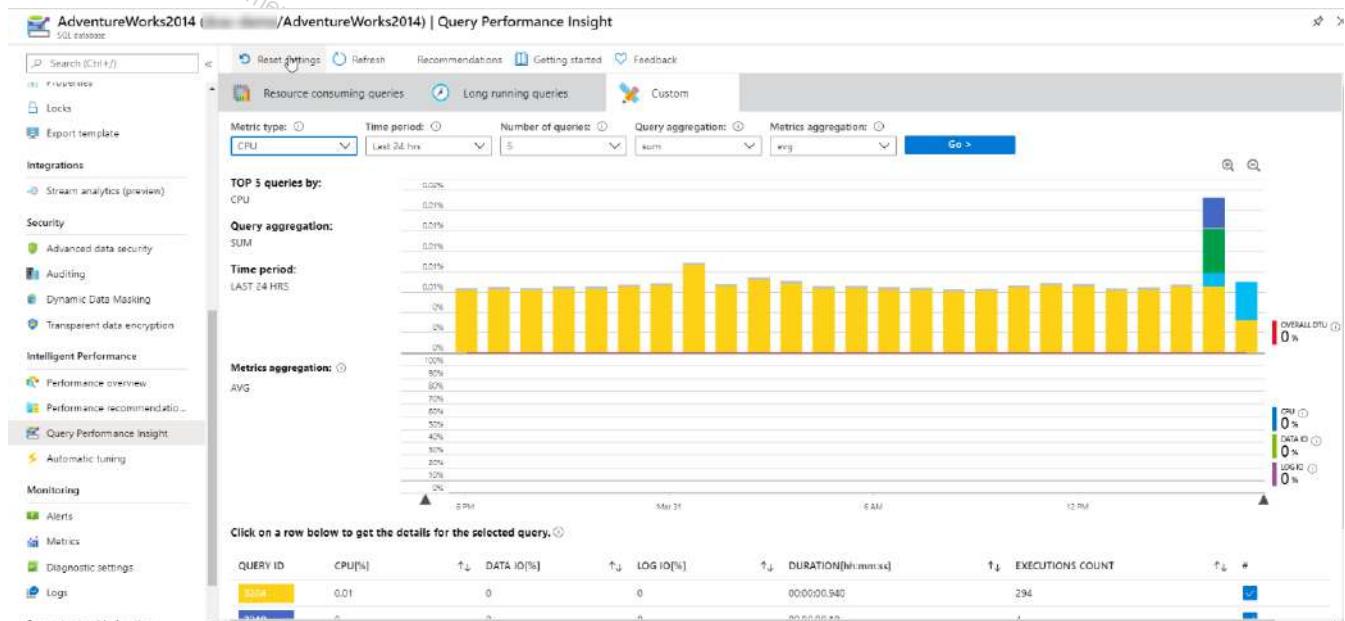


Switching to Long Running Queries, you can see a similar layout as before. In this case, the metrics are limited to the top five queries sorted by duration from the previous 24 hours and is a sum aggregation. In the grid below the graph, you can examine specific queries by clicking on the row.



By switching to the custom tab, there's a little more flexibility compared to the other two options.

Within this tab, we can further define how we wish to examine performance data. It offers us several drop-down menus that will drive the visual representation of the data. The key metrics are CPU, Log IO, Data IO, and memory. These metrics are the aspects of database performance, the upper limits of which are determined by the service tier and compute resources of your Azure SQL Database.



If we drill into an individual query, we'll be able to see the query ID and the query itself, as well as the query aggregation type and associated time period. Furthermore, the query ID also correlates to the query ID located within the Query Store. Metrics gleaned from Query Performance Insights can then be easily located within the Query Store itself for deeper analysis or possibly problem resolution if needed.



Query details

AdventureWorks2014 - Query ID 3204

[Settings](#) [Refresh](#)[Recommendations](#) [Query Text](#)

Query ID 3204:

```

1  ↴ |SELECT c.*,
2      |          i.object_id, i.unique_index_id, i.is_enabled,
3      |          i.change_tracking_state_desc, i.has_crawl_completed,
4      |          i.crawl_type_desc, i.crawl_start_date, crawl_end_date,
5      |          i.incremental_timestamp, i.stoplist_id, i.data_space_id,
6      |          i.property_list_id,
7      |          cast(OBJECTPROPERTYEX(i.object_id, 'TableFullTextMergeStatus') as
8      |          int) as merge_status,
9      |          cast(OBJECTPROPERTYEX(i.object_id, 'TableFulltextDocsProcessed') as
10     |          int) as docs_processed,

```

Details of Query ID 3204 (Query aggregation: sum) Last 24 hrs



100%

90%

80%

70%

60%

50%

40%

30%

20%

10%

0%

150ms

100ms

50ms

0ms

15

10

5

0

CPU FOR 3204 ⓘ

0.01 %

DATA IO FOR 3204 ⓘ

0 %

LOG IO FOR 3204 ⓘ

0 %

DURATION FOR 3204 ⓘ

37.6 ms

EXECUTION COUNT F...

11.88

6 PM

Mar 31

6 AM

12 PM

While Query Performance Insight doesn't show the query's execution plan, you can quickly identify that query, and use the information to extract the plan from the Query Store in your database.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which Intelligent Insights options provide SQL Insights?

- Log Analytics
- Azure Storage
- Event Hubs

Check Answers

Multiple choice

Which Performance Monitor counter reflects how long SQL Server expects to retain data in memory?

- Page Life Expectancy
- Processor Queue Length
- Paging File Usage

Check Answers

Multiple choice

What tool should you use to see the sizes of your SQL Server databases running in an Azure virtual machine?

- The SQL virtual machine resource provider
- Azure Monitor
- SQL Insights

Check Answers

Multiple choice

Which extended event target counts how many times each specified event occurs?

- The `etw_classic_sync_target` target
- The `event_file` target
- The `event_counter` target

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

The Azure platform allows you to use the Azure Monitor to collect baseline performance data about both PaaS and IaaS resources. Within Windows, the Performance Monitor allows you to collect detailed performance information about your SQL Server. Azure SQL Database includes additional detailed query performance information through Query Performance Insight, and provides advanced monitoring capabilities through extended events.

Having a solid understand of the baseline workload of your server and database are critical to understanding performance anomalies.

Now that you've reviewed this module, you should be able to:

- Understand methods to review potential performance issues
- Identify critical Azure metrics
- Collect metrics for an established baseline
- Use extended events for performance analysis
- Understand Azure SQL Insights

Introduction

One of the challenges administrators face in the cloud is balancing costs and performance. Both Azure and SQL Server provide many options for configuration to meet the needs of small and large workloads. Choosing the right storage and sizing your virtual machine are critical steps in meeting the performance needs of your applications and balancing cloud costs. Proper configuration of SQL Server resources like TempDB which can easily become a performance bottleneck, and Resource Governor, which can be used to manage multi-tenant workloads, are also important for properly maintaining your server performance.

Learning objectives

In this module, you will:

- Understand your options for configuration of Azure storage
- Learn how to configure TempDB data files in SQL Server
- Learn how to choose the right type of VM for SQL Server workloads
- Understand the use cases and configuration of Resource Governor in SQL Server

Explain how to optimize Azure storage for SQL Server virtual machines

Storage performance is a critical component of an I/O heavy application like a database engine. Azure offers a broad array of storage options and can even build your storage solution to meet your workload requirements.

Azure Storage is a highly scalable, secure storage platform that offers a range of solutions to meet the needs of many applications. Because the focus of this course is databases, you will learn about the aspects of blob storage that are applicable to SQL Server workloads, which are disk, file, and blob storage. Note that all the above types of storage support encryption at rest with either a Microsoft managed or a user-defined encryption key.

Blob Storage - Blob storage is what is known as object-based storage and includes cold, hot, and archive storage tiers. In a SQL Server environment, blob storage will typically be used for database backups, using SQL Server's back up to URL functionality.

File Storage - File storage is effectively a file share that can be mounted inside a virtual machine, without the need to set up any hardware. SQL Server can use File storage as a storage target for a failover cluster instance.

Disk Storage - Azure managed disks offer block storage that is presented to a virtual machine. These disks are managed just like a physical disk in an on-premises server, except that they are virtualized. There are several performance tiers within managed disks depending on your workload. This type of storage is the most commonly used type for SQL Server data and transaction log files.

Azure managed disks

Azure managed disks are block-level storage volumes that are presented to Azure Virtual Machines. Block level storage refers to raw volumes of storage that are created and can be treated as an individual hard drive. These block devices can be managed within the operating system, and the storage tier is not aware of the contents of the disk. The alternative to block storage is object storage, where files and their metadata are stored on the underlying storage system. Azure Blob Storage is an example of an object storage model. While object storage works well for many modern development solutions, most workloads running in virtual machines will use block storage.

The configuration of your managed disks is important to the performance of your SQL Server workloads. If you are moving from an on-premises environment, it is important to capture metrics like **average disk seconds/read** and **average disk seconds/write** from Performance Monitor as detailed earlier. Another metric to capture is the I/O Operations per Second, which can be captured using the **SQL Server: Resource Pool Stats Disk Read and Write IO/sec** counters, which show you how many IOPs SQL Server is serving at its peak. It is important to understand your workloads. You will want to design your storage and virtual machine to meet the needs of those workload peaks without incurring significant latency. Note that each Azure Virtual Machine type has a limit on IOPs.

Azure managed disks come in four types:

Ultra disk - Ultra disks support high-IO workloads for mission critical databases with low latency.

Premium SSD - Premium SSD disks are high-throughput and low latency and can meet the needs of most database workloads running in the cloud.

Standard SSD - Standard SSDs are designed for lightly used dev/test workloads or web servers that do a small amount of IO, and require predictable latency.

Standard HDD - Standard HDDs are suitable for backups and file storage that is infrequently accessed.

Typically, production SQL Server workloads will use either Ultra disk or Premium SSD, or some combination of the two. Ultra disks are typically used where you are looking for submillisecond latency in response time. Premium SSDs typically have single digit millisecond response time, but have lower costs, and more flexibility in design. Premium SSDs also support read-caching, which can benefit read-heavy database workloads by reducing the number of trips to the disk. The read cache is stored on the local SSD (the D:\ drive on Windows or /dev/sdb1/ on Linux) which can help reduce the number of round trips to the actual disk.

Striping disks for maximum throughput

One of the ways to get more performance and volume out of Azure disks is to stripe your data across multiple disks. This technique does not apply to Ultra disk, as you can scale IOPs, throughput, and maximum size independently on a single disk. However, with Premium SSDs it can be beneficial to scale both IOPs and storage volume. In order to stripe disks in Windows, you simply add the number of disks you would like to the VM, and then create a pool using Storage Spaces in Windows. Don't configure any redundancy for your pool (which would limit your performance) as the redundancy is provided by the Azure framework, which keeps three copies of all disks in synchronous replication to protect against a disk failure. When you create a pool, your pool has the sum of the IOPs and the sum of the volume of all the disks in your pool. For example, if you used 10 P30 disks that are each one TB and have 5000 IOPs per disk, you would have a 10-TB volume with 50,000 IOPs available.

SQL Server storage configuration best practices

There are few recommendations for best practices for SQL Server on Azure VMs and their storage configuration:

- Create a separate volume for data and transaction log files
- Enable read caching on the data file volume
- Do not enable any caching on the log file volume
- Plan for an additional 20% of IOPs and throughput when building your storage for your VM to handle workload peaks
- Use the D: drive (the locally attached SSD) for TempDB files because TempDB is recreated upon server restart, so there is no risk of data loss
- Enable instant file initialization to reduce the impact of file-growth activities.
- Move trace file and error log directories to data disks
- For workloads requiring storage latency under one millisecond, consider using Ultra disk over Premium SSD.

Azure Virtual Machine resource provider

One way to reduce the complexity of building storage for your SQL Server on an Azure Virtual Machine is to use the SQL Server templates in the Azure Marketplace, which allow you to configure your storage as part of your deployment as shown below. You can configure the IOPs as needed and the template will perform the work of creating your storage spaces pools within Windows.

Configure storage



Storage optimization ⓘ

General

Transactional processing

Data warehousing

Data storage

These disks will be attached to your virtual machine as data disks and will be stored in storage as page blobs.

Data drive location * ⓘ

F:\data

Disk type * ⓘ

Premium SSD

Disk type

Size (GiB)

Max IOPS

Max throughput

Number of disks

1024 GiB, Premium SSD (...

1024

5000

200

1

1024 GiB, 5000 IOPS, 200 MB/s

Log storage

Transaction logs are a critical component of the database as they record all transactions and database modifications made by each transaction.

Shared drive space * ⓘ

Use a separate drive for lo...

Log drive location * ⓘ

G:\log

Disk type * ⓘ

Premium SSD

Disk type

Size (GiB)

Max IOPS

Max throughput

Number of disks

1024 GiB, Premium SSD (...

1024

5000

200

1

1024 GiB, 5000 IOPS, 200 MB/s

TempDb storage

The tempDb system database is a global resource that is available to all users connected to the instance of SQL Server. It is used to store temporary user objects and internal objects created by the database engine.

Shared drive space * ⓘ

Use local SSD drive

TempDb drive location * ⓘ

D:\tempDb

This resource provider also supports adding TempDB to the local SSD drive and creates a scheduled task to create the folder on startup.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Describe virtual machine resizing

There are many size options for Azure Virtual Machines. For SQL Server workloads the main characteristics to look for are the amount of memory available, and the number of input and output operations (IOPs) the virtual machine can perform.

Using constrained cores

Typically SQL Server is licensed by the core and Azure will use a fixed ratio of CPU cores to memory. However, you may have workloads that require large amounts of memory, but do not require the default amount of allocated CPUs to get that amount of memory. In these cases it may be useful to use Azure's constrained cores.

With constrained cores, you can reduce the cost of software licensing while still getting the full amount of memory, storage, and I/O bandwidth. This is good for database workloads that are not CPU-intensive and can benefit from high memory, storage, and I/O bandwidth, while using a constrained vCPU count.

Using general purpose virtual machines

Most SQL Server production workloads will run on the general purpose or memory-optimized families of Azure Virtual Machines. Larger workloads requiring more memory and/or CPU resources will land in memory-optimized virtual machines, but many production applications can run comfortably on general purpose virtual machines.

Resizing virtual machines

Azure supports resizing your virtual machine. This operation does require a restart; however, restarting a virtual machine is typically a fast process. In some cases, depending on what virtual machine type you are switching to and from, you may need to deallocate your virtual machine and then resize. This operation does extend the duration of the outage but should not take more than a few minutes.

Optimize database storage

To optimize database storage, you should consider proportional fill and tempdb configuration.

What is proportional fill?

If you are inserting one gigabyte of data into a SQL Server database with two data files, you would expect the size of each of your data files to increase by roughly 512 megabytes. However, this equal growth is not necessarily the case, as SQL Server will insert data into data files in different volumes based on the size of the data file. In the above example, if your data files were both two gigabytes in size, you would expect the even distribution of data. However, if one of your data files was 10 gigabytes, and the other was one gigabyte, roughly 900 MB would go into the ten-gigabyte file, and 100 MB into the one-gigabyte file. While this behavior occurs in any database, with the write-intensive nature of tempdb, an uneven write pattern could cause a bottleneck on the largest file, as more of the writes would happen there.

Tempdb configuration in SQL Server

SQL Server 2016 changed this behavior, by detecting the number of CPUs available at setup, and configuring the proper number of files, up to 8, and sizing the data files evenly. Additionally, the behaviors of trace flags 1117 and 1118 are built in to the database engine, but only for tempdb. For tempdb heavy workloads, there may be benefits to increasing the number of tempdb files beyond eight, to the number of CPUs on your machine.

SQL Server uses tempdb for much more than storing user-defined temporary tables. Work tables that are used to store intermediate query results, sorting operations, and the version store for row versioning are among just a few of the uses for tempdb. Because of this utilization, it is important both to place tempdb on the lowest latency storage possible, and to properly configure its data files.

Prior to SQL Server 2016, tempdb defaulted to having only one data file. This single file meant that there could be contention for multiple processes trying to access system pages of the tempdb database. One common solution to this contention problem was to enable trace flag 1118, which changed the way extents were allocated. Another common best-practice recommendation was to create multiple tempdb data files. Because SQL Server uses a proportional fill algorithm for databases with multiple data files, it was also important to ensure that those files were the same size and grew at the same rate. To support this many DBAs used trace flag 1117, which forced all databases with multiple data files to grow them at the same rate.

Control SQL Server resources

While some SQL Servers or Azure SQL managed instances only support one application's databases (this configuration is commonly seen in mission critical applications), many servers support databases for multiple applications with differing performance requirements and different peak workload cycles. Balancing these differing requirements can be challenging to the administrator. One of the ways to balance server resources is to use Resource Governor, which was introduced to SQL Server 2008.

Resource Governor is a feature in SQL Server and Azure SQL managed instance that allows you to granularly control how much CPU, physical IO, and memory resources can be used by an incoming request from an application. Resource Governor is enabled at the instance level and allows you to define how connections are treated by using a classifier function, which subdivides sessions into workload group. Each workload group is configured to use a specific pool of system resources.

Resource pools

A resource pool represents physical resources available on the server. SQL Server always has two pools, default and internal, even when Resource Governor is not enabled. The internal pool is used by critical SQL Server functions and cannot be restricted. The default pool, and any resource pools you explicitly define, can be configured with limits on the resources it can use. You can specify the following limits for each non-internal pool:

- Min/Max CPU percent
- Cap of CPU percent
- Min/Max memory percent
- NUMA node affinity
- Min/Max IOPs per volume

NOTE: Changes to a pool only affect new sessions, not existing ones. A change to a pool will not help you restrict the resources of a long-running process. The exception is external pools used in conjunction with SQL Server Machine Learning Services, which are external to SQL Server and can be limited by a pool change.

With the exception of min/max CPU percent, all of the other resource pool settings represent hard limits and cannot be exceeded. Min/Max CPU percentage will only apply when there is CPU contention. For example, if you have a maximum of 70%, if there is available CPU cycles the workload may use up to 100%. If there are other workloads running, the workload will be restricted to 70%.

Workload group

A workload group is a container for session requests based on their classification by the classifier function. Like resource pools there are two built-in groups, default and internal, and each workload group can only belong to one resource pool. However, a resource pool can host multiple workload groups. All connections go into the default workload group, unless they passed into another user-defined group by the classifier function. And by default, the default workload group uses the resources assigned to the default resource pool.

Classifier function

The classifier function is run at the time a connection is established to the SQL Server instance and classifies each connection into a given workload group. If the function returns a NULL, default, or the name of the non-existent workload group the session is transferred into the default workload group. Since the classifier is run at every connection, it should be tested for efficiency. The following image shows a sample classifier function that classifies users based on their user name.

```
CREATE FUNCTION dbo.RGClassifier()
RETURNS SYSNAME
WITH SCHEMABINDING
AS
BEGIN
DECLARE @workloadGroup AS SYSNAME
IF(SUSER_NAME() = 'ReportUser')
SET @workloadGroup = 'ReportServerGroup'
ELSE IF (SUSER_NAME() = 'PrimaryUser')
SET @workloadGroup = 'PrimaryServerGroup'
ELSE
SET @workloadGroup = 'default'
RETURN @workloadGroup
END
```

You can increase the complexity of the function definition shown in the example, but you should verify that the more complex function doesn't impact the user login performance.

Resource Governor use cases

Resource Governor is used primarily in multi-tenant scenarios where a group of databases share a single SQL Server instance, and performance needs to be kept consistent for all users of the server. You can also use Resource Governor to limit the resources used by maintenance operations like consistency checks and index rebuilds, to try to guarantee sufficient resources for user queries during your maintenance windows.

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which type of storage should be used in conjunction with Azure VMs for SQL Server data files?

- Table storage
- Blob storage
- Disk storage

Check Answers

Multiple choice

Which of the following can be limited using Resource Governor?

- Buffer pool allocation
- Write IOPs
- Recompilation

Check Answers

Multiple choice

Which is an option from the SQL Server Resource Provider for Azure VMs?

- Storage configuration
- Changing max degree of parallelism
- Maintenance plan

Check Answers

Summary

Proper storage configuration is important to the performance of your Azure Virtual Machines. SQL Server should run on premium disk storage or ultra disk for optimal performance. You can use the Resource Provider for SQL Server to automate the creation of your storage for you SQL Servers on Azure Virtual Machines. You can use the Resource Governor to manage differing workloads in the same SQL Server.

Now that you've reviewed this module, you should be able to:

- Understand your options for configuration of Azure storage
- Describe how to configure tempdb data files in SQL Server
- Describe how to choose the right type of virtual machine for SQL Server workloads
- Understand the use cases and configuration of Resource Governor in SQL Server

Introduction

In recent versions of SQL Server, Microsoft has moved more configuration options to the database level, giving you more granularity in how your databases behave. Along with those options, they have introduced intelligent query processing features that allow the query optimizer to make better choices.

Even when your database is in the cloud, on-going performance-related maintenance tasks are critical to the overall success of your applications. Whether it's a SQL Server instance in an Azure Virtual Machine or Azure SQL Database, you need to ensure your statistics are current, and your indexes are well organized.

Learning objectives

In this module, you will:

- Understand database scoped configuration options
- Understand maintenance tasks related to indexing and statistics
- Understand the features of Intelligent Query Processing
- Explore the automatic tuning feature in Azure

Explore database maintenance checks

The query optimizer utilizes statistical information from the indexes to attempt to build the most optimal execution plan.

Within Azure SQL maintenance tasks such as backups and integrity checks are handled for you, and while you may be able to get away with automatic updates keeping your statistics up-to-date, sometimes it's not enough.

Having healthy indexes and statistics will ensure that any given plan will perform at optimal efficiency. Index maintenance should be performed regularly as data in your databases changes over time. You could change your index maintenance strategy based on the frequency of modifications to your data.

Rebuild and reorganize

Index fragmentation occurs when logical ordering within index pages doesn't match the physical ordering. Pages can come out of order during routine data modification statements such as `UPDATE`, `DELETE`, and `INSERT`.

Fragmentation can introduce performance issues because of the extra I/O that is required to locate the data that is being referenced by the pointers within the index pages.

As data is inserted, updated, and deleted from indexes the logical ordering in the index will no longer match the physical ordering inside of the pages, and between the pages, making up the indexes. Also, over time the data modifications can cause the data to become scattered or fragmented in the database. Fragmentation can degrade query performance when the database engine needs to read extra pages in order to locate needed data.

A reorganization of an index is an online operation that will defrag the leaf level of the index (both clustered and nonclustered). This defragmentation process will physically reorder the leaf-level pages to match the logical order of the nodes from left to right. During this process, the index pages are also compacted based on the configured `fillfactor` value.

A rebuild can be either online or offline depending on the command executed or the edition of SQL Server being utilized. An offline rebuild process will drop and re-create the index itself. If you can do so online, a new index will be built in parallel to the existing index. Once the new index has been built, the existing one will be dropped and then the new one will be renamed to match the old index name. Keep in mind that the online version will require more space as the new index is built in parallel to the existing index.

The common guidance for index maintenance is:

- **> 5% but < 30%** - Reorganize the index
- **> 30%** - Rebuild the index

Use these numbers as general recommendations. Depending on your workload and data, you may need to be more assertive, or in some cases you may be able to defer index maintenance for databases that mostly perform queries that seek specific pages.

The SQL Server and Azure SQL platforms offer DMVs that allow you to detect fragmentation in your objects. The most commonly used DMVs for this purpose are `sys.dm_db_index_physical_stats` for b-tree indexes, and `sys.dm_db_column_store_row_group_physical_stats` for columnstore indexes.

One other thing to note is that index rebuilds cause the statistics on the index to be updated, which can further help performance. Index reorganization doesn't update statistics.

Microsoft introduced resumable rebuild index operations with SQL Server 2017. Resumable rebuild index operations option provides more flexibility in controlling how much time a rebuild operation might impose on a given instance. With SQL Server 2019, the ability to control an associated maximum degree of parallelism was introduced further providing more granular control to database administrators.

Statistics

When doing performance tuning in Azure SQL, understanding the importance of statistics is critical.

Statistics are stored in the user database as binary large objects (blobs). These blobs contain statistical information about the distribution of data values in one or more columns of a table or indexed view.

Statistics contain information about the distribution of data values within a column. The query optimizer uses column and index statistics in order to determine cardinality, which is the number of rows a query is expected to return.

Cardinality estimates are then used by the query optimizer to generate the execution plan. Cardinality estimates also help the optimizer determine what type of operation (for example, index seek or scan) to use to retrieve the data requested.

To see the list of user defined statistics with the last updated date, run the query below:

```
SELECT sp.stats_id,
       name,
       last_updated,
       rows,
       rows_sampled
  FROM sys.stats
  CROSS APPLY sys.dm_db_stats_properties(object_id, stats_id) AS sp
 WHERE user_created = 1
```

Create statistics

When you have `AUTO_CREATE_STATISTICS` option to `ON`, the query optimizer creates statistics on the indexed column by default. The query optimizer also creates statistics for single columns in query predicates.

These methods provide high-quality query plans for most queries. At times, you may need to create more statistics using `CREATE STATISTICS` statement to improve specific query plans.

It's recommended to keep the `AUTO_CREATE_STATISTICS` option enabled as it will allow the query optimizer to create statistics for query predicate columns automatically.

Whenever you encounter the following situations, consider creating statistics:

- The Database Engine Tuning Advisor suggests creating statistics
- The query predicate contains multiple columns that aren't already in the same index
- The query selects from a subset of data
- The query has missing statistics

Maintenance tasks automation

Azure SQL provides native tools to perform database maintenance tasks for automation purposes. Different tools are available depending on the platform where the database is running.

SQL Server on an Azure Virtual Machine

You have access to scheduling services such as the SQL Agent or the Windows Task Scheduler. These automation tools can help keeping the amount of fragmentation within indexes to a minimum. With larger databases, a balance between a rebuild and a reorganization of indexes must be found to ensure optimal performance. The flexibility provided by SQL Agent or Task Scheduler allows you to run custom jobs.

Azure SQL Database

Due to the nature of Azure SQL Database, you don't have access to SQL Server Agent nor Windows Task Scheduler. Without these services, index maintenance must be created using other methods. There are three ways to manage maintenance operations for SQL Database:

- Azure Automation runbooks
- SQL Agent Job from SQL Server in an Azure Virtual Machine (remote call)
- Azure SQL elastic jobs

Azure SQL Managed Instance

As with SQL Server on an Azure Virtual Machine, you can schedule jobs on a SQL Managed Instance through SQL Server Agent. Using SQL Server Agent provides flexibility to execute code designed to reduce fragmentation within the indexes in the database.

Describe database scoped configuration options

SQL Server has always had configuration options that were set at the database level. For example, the recovery model has always been a database setting, but as more complex features have been introduced to the database, more options have been added. Many of these options are tied to the compatibility level of the database, which is itself a database level configuration option. Database configuration options break down into two groups, with a minor difference:

- Options configured by the `ALTER DATABASE SCOPED CONFIGURATION` syntax in T-SQL
- Options configured by the `ALTER DATABASE` syntax in T-SQL

There's no significance to the different ways to set these options. Options that are set using `ALTER DATABASE` include:

- **Database recovery model** – Whether the database is in full or simple recovery model
- **Automatic tuning option** – Whether to enable the force last good plan
- **Auto create and update statistics** – Allows the database to create and update statistics and allows for the option of asynchronous statistics updates
- **Query store options** – The Query Store options are configured here
- **Snapshot isolation** – You can configure snapshot isolation and read committed snapshot isolation

The above settings are a subset of the configurable options.

Many options previously configured on the server can now be configured at the database level. Some of the options include:

- **Maximum Degree of Parallelism** – Allows for a database to configure its own MaxDOP setting and override the server's setting.
- **Legacy Cardinality Estimation** – Allows for the database to use the older cardinality estimator. Some queries may have degraded performance under the newer cardinality estimator, and may benefit from it. You should note that if you use this option with a newer compatibility level, you can still get the benefits of Intelligent Query Processing in compatibility level 140 or 150.
- **Last Query Plan Stats** – Allows you to capture the values of the last actual execution plan for a query. This feature is only active in compatibility level 150.
- **Optimize for Ad Hoc Workloads** – Uses the optimizer to store a stub query plan in the plan cache. This can help reduce the size of the plan cache for workloads that have numerous single use queries.

Database compatibility level

Each database has its own compatibility level, which controls the behavior of the query optimizer for that database.

You can manage this setting when upgrading SQL Server to ensure that your queries have similar execution plans to the older version.

Microsoft will support running on an older compatibility level for an extended period. You should upgrade to a newer compatibility level if possible, as many of the new features in Intelligent Query Processing are only available in compatibility level 140 or 150.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Describe automatic tuning

Automatic tuning is a monitoring and analysis feature that continuously learns about your workload and identifies potential issues and improvements.

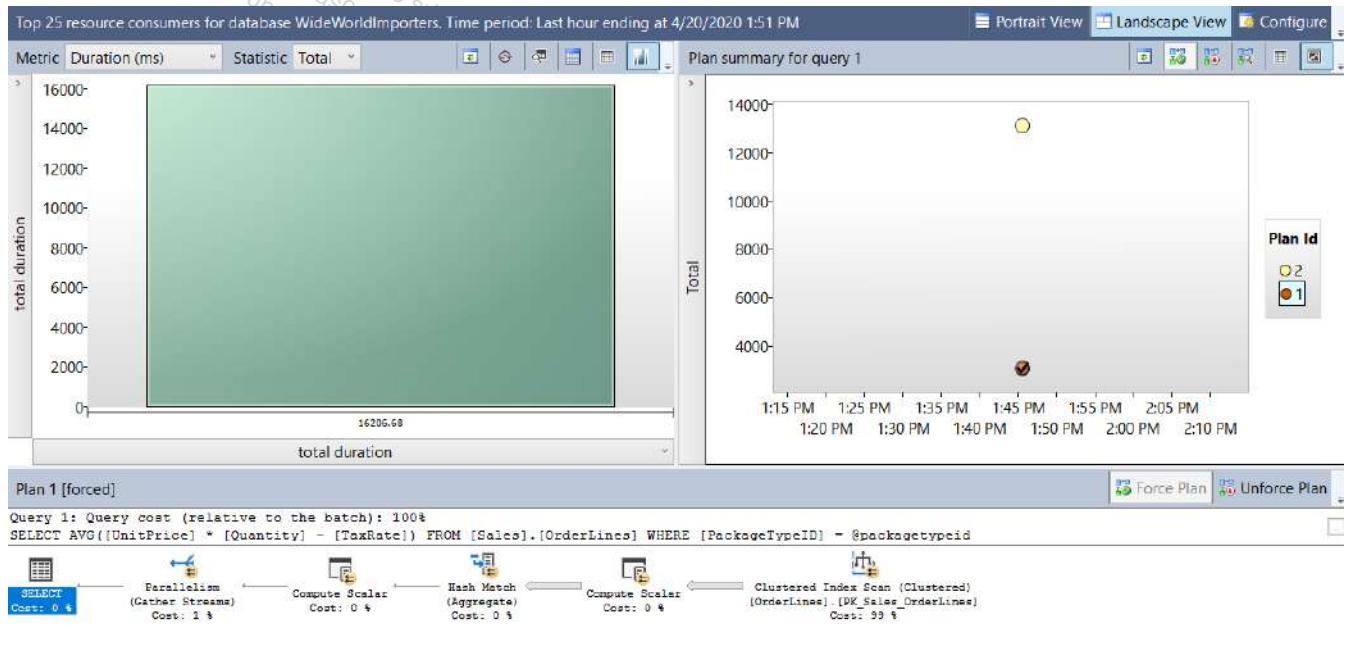
The automatic tuning recommendations are based on the data collected from Query Store. Execution plans evolve over time due to schema changes, index modifications, or changes to the data that cause updates to the statistics. This evolution can cause queries to perform poorly as the execution plan no longer meets the demands of the given query.

Furthermore, automatic tuning allows for the gathering and applying machine learning services against performance metrics to provide suggested improvements or even allow for self-correction.

Whether on-premises or in the cloud, automatic tuning allows you to identify issues caused by query execution plan regression. Additionally, in Azure SQL Database you can improve query performance by index tuning. Azure SQL Database automatic tuning can identify indexes that should be added or even removed from the database to enhance query performance.

Automatic plan correction

With the help of the Query Store data, the database engine can determine when query execution plans have regressed in performance. While you can manually identify a regressed plan through the user interface, the Query Store also provides the option to notify you automatically.



In the example above, you can see a check mark on **Plan ID 1**, which means that the plan has been forced. After the feature is enabled, the database engine will automatically force any recommended query execution plan, when:

- The previous plan had a higher error rate than the recommended plan
- The estimated CPU gain was greater than 10 seconds
- The force plan has performed better than the previous one

The plan will revert back to the last known good plan after 15 executions of the query.

When plan forcing occurs automatically, the database engine will apply the last known good plan and will also continue to monitor query execution plan performance. If the forced plan doesn't perform better than the previous plan, it will be then unforced and force a new plan to be compiled. If the forced plan continues to outperform the previously bad plan, it will remain forced until such time as a recompile occurs.

You can enable automatic plan correction via a T-SQL query, as shown below. The Query Store must be enabled and must be in Read-Write mode for the command to succeed. If either of those two criteria aren't met, the ALTER statement will fail.

```
ALTER DATABASE [WideworldImporters] SET AUTOMATIC_TUNING (FORCE_LAST_GOOD_PLAN = ON);
```

You can examine the automatic tuning recommendations through a dynamic management view (DMV), `sys.dm_db_tuning_recommendations`, which is available in SQL Server 2017 or higher and is also available in Azure SQL Database solutions. This DMV provides information such as reasons as to why the recommendation was provided, the type of recommendation, and the state of the recommendation. To confirm that automatic tuning is enabled for a database, check the view `sys.database_automatic_tuning_options`.

Automatic index management

Azure SQL Database can perform automatic index tuning. Over time, the database will learn about existing workloads and provide recommendations on adding or removing indexes in order to provide better performance. Like forcing improved query plans, the database can be configured to allow for automatic index creation or removal depending on existing index performance, as shown below:

Azure SQL Database built-in intelligence automatically tunes your databases to optimize performance. Click here to learn more about automatic tuning.

Inherit from: Server Azure defaults Don't inherit

Information The database is inheriting automatic tuning configuration from the server. You can set the configuration to be inherited by going to: [Server tuning settings](#)

Configure the automatic tuning options [\(i\)](#)

Option	Desired state	Current state
FORCE PLAN	ON OFF INHERIT	ON Inherited from server
CREATE INDEX	ON OFF INHERIT	OFF Inherited from server
DROP INDEX	ON OFF INHERIT	OFF Inherited from server

When enabled, the **Performance Recommendations** page will identify indexes that can be created or dropped depending on query performance. Remember this feature isn't available for on-premises databases and only available for Azure SQL Database.

Alternatively, use the following query to see the automatic tuning features enabled in your database:

```
SELECT name,
       desired_state_desc,
       actual_state_desc,
```

```
reason_desc  
FROM sys.database_automatic_tuning_options
```

Creating new indexes can consume resources, and the timing of the index creations is critical to ensure no negative effect is felt on your workloads.

Azure SQL Database will monitor the resources required to implement new indexes to avoid causing performance degradation. The tuning action is postponed until the available resources are available, for example if resources are required for existing workloads and not available for creating an index.

Monitoring ensures any action taken won't harm performance. If an index is dropped and query performance noticeably degrades, the recently dropped index will be automatically recreated.

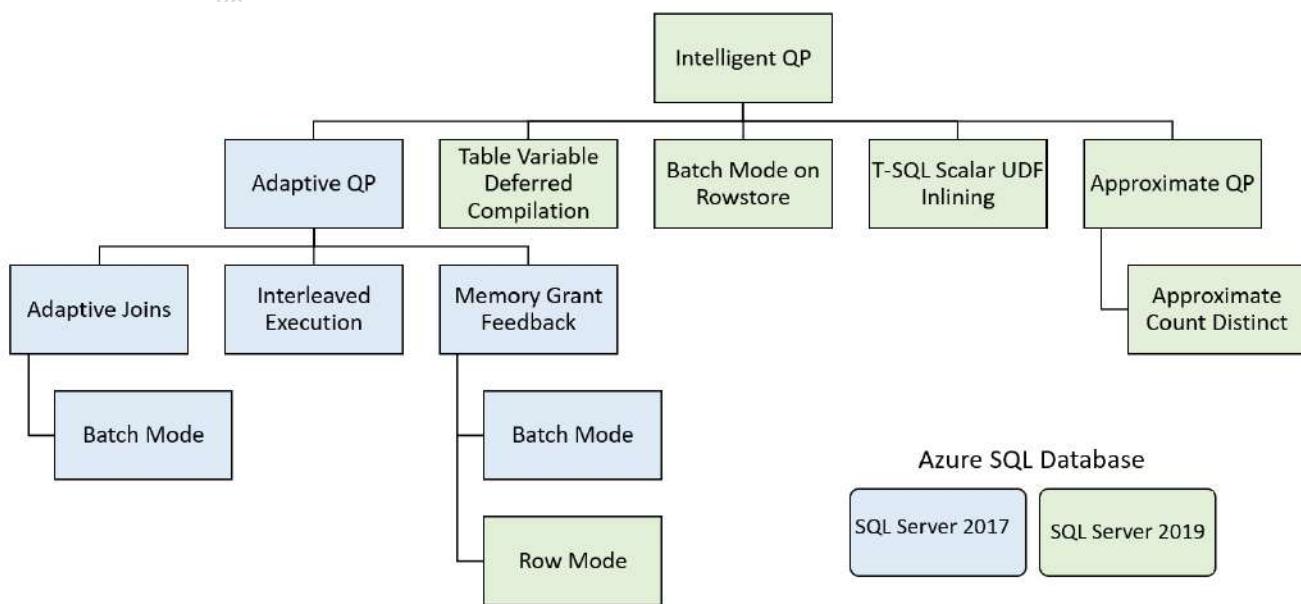
This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Describe intelligent query processing

In SQL Server 2017 and 2019, and with Azure SQL, Microsoft has introduced many new features into compatibility levels 140 and 150. Many of these features correct what were formerly anti-patterns like using user defined scalar value functions and using table variables.

These features break down into a few families of features:



Intelligent query processing includes features that improve existing workload performance with minimal implementation effort.

To make workloads automatically eligible for intelligent query processing, change the applicable database compatibility level to 150. For example:

```
ALTER DATABASE [wideworldImportersDW] SET COMPATIBILITY_LEVEL = 150;
```

Adaptive query processing

Adaptive query processing includes many options that make query processing more dynamic, based on the execution context of a query. These options include several features that enhance the processing of queries.

- **Adaptive Joins** – the database engine defers choice of join between hash and nested loops based in the number of rows going into the join. Adaptive joins currently only work in batch execution mode.
- **Interleaved Execution** – Currently this feature supports multi-statement table-valued functions (MSTVF). Prior to SQL Server 2017, MSTVFs used a fixed row estimate of either one or 100 rows, depending on the version SQL Server. This estimate could lead to suboptimal query plans if the function returned many more rows. An actual row count is generated from the MSTVF before the rest of the plan is compiled with interleaved execution.
- **Memory Grant Feedback** – SQL Server generates a memory grant in the initial plan of the query, based on row count estimates from statistics. Severe data skew could lead to either over- or under-estimates of row counts, which can cause over-grants of memory that decrease concurrency, or under-grants, which can cause

the query to spill data to tempdb. With Memory Grant Feedback, SQL Server detects these conditions and decreases or increases the amount of memory granted to the query to either avoid the spill or overallocation.

These features are all automatically enabled under compatibility mode 150 and require no other changes to enable.

Table variable deferred compilation

Like MSTVF, table variables in SQL Server execution plans carry a fixed row count estimate of one row. Much like MSTVF, this fixed estimate led to poor performance when the variable had a much larger row count than expected. With SQL Server 2019, table variables are now analyzed and have an actual row count. Deferred compilation is similar in nature to interleaved execution for MSTVF, except that it's performed at the first compilation of the query rather than dynamically within the execution plan.

Batch mode on row store

Batch execution mode allows data to be processed in batches instead of row by row. Queries that incur significant CPU costs for calculations and aggregations will see the largest benefit from this processing model. By separating batch processing and columnstore indexes, more workloads can benefit from batch mode processing.

Scalar user-defined function inlining

In older versions of SQL Server, scalar functions performed poorly for several reasons. Scalar functions were executed iteratively, effectively processing one row at a time. They didn't have proper cost estimation in an execution plan, and they didn't allow parallelism in a query plan. With user-defined function inlining, these functions are transformed into scalar subqueries in place of the user-defined function operator in the execution plan. This transformation can lead to significant gains in performance for queries that involve scalar function calls.

Approximate count distinct

A common data warehouse query pattern is to execute a distinct count of orders or users. This query pattern can be expensive against a large table. Approximate count distinct introduces a much faster approach to gathering a distinct count by grouping rows. This function guarantees a 2% error rate with a 97% confidence interval.

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which platform supports automatic index management?

- Azure SQL Managed Instance
- Azure SQL Database
- SQL Server in an Azure VM

Check Answers

Multiple choice

Which statistics option allows statistics to be updated while a query executes against the object the statistics are based on?

- Auto Create Incremental Statistics
- Auto Create Statistics
- Auto Update Statistics Asynchronously

Check Answers

Multiple choice

Which DMV shows the status of a plan updated by automatic tuning?

- sys.dm_db_tuning_recommendations
- sys.dm_db_automatic_tuning_options
- sys.query_store_query

Check Answers

Multiple choice

Which component of resource governor allows you to configure limits on system resources?

- Workload groups
- Classifier functions
- Resource pools

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

Azure SQL has included many features in recent releases to improve performance. Many of these features can be enabled at the individual database level, and may also be controlled using the compatibility level of the database.

Maintaining indexes and statistics can help provide consistent performance for your queries. Additionally, Azure SQL Database automates the creation of new indexes and the removal of unused indexes.

Now that you've reviewed this module, you should be able to:

- Understand database scoped configuration options
- Understand maintenance tasks related to indexing and statistics
- Understand the features of Intelligent Query Processing
- Explore the automatic tuning feature in Azure

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Introduction

The most important skill you should acquire in database performance tuning is being able to read and understand query execution plans. The plans explain the behavior of the database engine as it executes queries and retrieves the results.

Query Store helps you quickly identify your most expensive queries, and find any changes in performance. The Query Store provides a powerful data collection including the automation of query plan and execution runtime. SQL Server and Azure SQL provide locking and blocking in order to manage concurrency and ensure data consistency. Finally, you can adjust the isolation levels in SQL Server to help manage concurrency.

Learning objectives:

In this module, you will:

- Compare the different types of execution plans
- Understand the purpose and benefits of the Query Store
- Investigate the available reports and data in the Query Store
- Understand how blocking and locking work in the SQL Server database engine

Understand query plans

It's helpful to have a basic understanding of how database optimizers work before taking a deeper dive into execution plan details. SQL Server uses what is known as cost-based query optimizer. The query optimizer calculates a cost for multiple possible plans based on the statistics it has on the columns being utilized, and the possible indexes that can be used for each operation in each query plan. Based on this information, it comes up with a total cost for each plan. Some complex queries can have thousands of possible execution plans. The optimizer doesn't evaluate every possible plan, but uses heuristics to determine plans that are likely to have good performance. The optimizer will then choose the lowest cost plan of all the plans evaluated for a given query.

Because the query optimizer is cost-based, it's important that it has good inputs for decision making. The statistics SQL Server uses to track the distribution of data in columns and indexes need be kept up to date, or it can cause suboptimal execution plans to be generated. SQL Server automatically updates its statistics as data changes in a table; however, more frequent updates may be needed for rapidly changing data. The engine uses many factors when building a plan including compatibility level of the database, row estimates based on statistics and available indexes.

When a user submits a query to the database engine, the following process happens:

1. The query is parsed for proper syntax and a parse tree of database objects is generated if the syntax is correct.
2. The parse tree from Step 1 is taken as input to a database engine component called the *Algebrizer* for binding. This step validates that columns and objects in the query exist and identifies the data types that are being processed for a given query. This step outputs a query processor tree, which is in the input for step 3.
3. Because query optimization is a relatively expensive process in terms of CPU consumption, the database engine caches execution plans in a special area of memory called the plan cache. If a plan for a given query already exists, that plan is retrieved from the cache. The queries whose plans are stored in cache will each have a hash value generated based on the T-SQL in the query. This value is referred to as the *query_hash*. The engine will generate a *query_hash* for the current query and then look to see if it matches any existing queries in the plan cache.
4. If the plan doesn't exist, the Query Optimizer then uses its cost-based optimizer to generate several execution plan options based on the statistics about the columns, tables, and indexes that are used in the query, as described above. The output of this step is a query execution plan.
5. The query is then executed using an execution plan that is pulled from the plan cache, or a new plan generated in step 4. The output of this step is the results of your query.

NOTE: To learn more about how the query processor works, see [Query Processing Architecture Guide](#)

Let's look at an example. Consider the following query:

```
SELECT orderdate,
       AVG(salesAmount)
  FROM FactResellersales
 WHERE ShipDate = '2013-07-07'
 GROUP BY orderdate;
```

In this example SQL Server will check for the existence of the *OrderDate*, *ShipDate*, and *SalesAmount* columns in the table *FactResellerSales*. If those columns exist, it will then generate a hash value for the query, and examine

the plan cache for a matching hash value. If there's plan for a query with a matching hash the engine will try to reuse that plan. If there's no plan with a matching hash, it will examine the statistics it has available on the *OrderDate* and *ShipDate* columns. The `WHERE` clause referencing the *ShipDate* column is what is known as the predicate in this query. If there's a nonclustered index that includes the *ShipDate* column SQL Server will most likely include that in the plan, if the costs are lower than retrieving data from the clustered index. The optimizer will then choose the lowest cost plan of the available plans and execute the query.

Query plans combine a series of relational operators to retrieve the data, and also capture information about the data such as estimated row counts. Another element of the execution plan is the memory required to perform operations such as joining or sorting data. The memory needed by the query is called the memory grant. The memory grant is a good example of the importance of statistics. If SQL Server thinks an operator is going to return 10,000,000 rows, when it's only returning 100, a much larger amount of memory is granted to the query. A memory grant that is larger than necessary can cause a twofold problem. First, the query may encounter a `RESOURCE_SEMAPHORE` wait, which indicates that query is waiting for SQL Server to allocate it a large amount of memory. SQL Server defaults to waiting for 25 times the cost of the query (in seconds) before executing, up to 24 hours. Second, when the query is executed, if there isn't enough memory available, the query will spill to tempdb, which is much slower than operating in memory.

The execution plan also stores other metadata about the query, including, but not limited to, the database compatibility level, the degree of parallelism of the query, and the parameters that are supplied if the query is parameterized.

Query plans can be viewed either in a graphical representation or in a text-based format. The text-based options are invoked with `SET` commands and apply only to the current connection. Text-based plans can be viewed anywhere you can run T-SQL queries.

Most DBAs prefer to look at plans graphically, because a graphical plan allows you to see the plan as a whole, including what's called the *shape* of the plan, easily. There are several ways you can view and save graphical query plans. The most common tool used for this purpose is SQL Server Management Studio, but estimated plans can also be viewed in Azure Data Studio. There are also third-party tools that support viewing graphical execution plans.

There are three different types of execution plans that can be viewed.

Estimated Execution Plan

This type is the execution plan as generated by the query optimizer. The metadata and size of query memory grant are based on estimates from the statistics as they exist in the database at the time of query compilation. To see a text-based estimated plan run the command `SET SHOWPLAN_ALL ON` before running the query. When you run the query, you'll see the steps of the execution plan, but the query will NOT be executed, and you won't see any results. The `SET` option will stay in effect until you set it OFF.

Actual Execution Plan

This type is same plan as the estimated plan; however this plan also contains the execution context for the query, which includes the estimated and actual row counts, any execution warnings, the actual degree of parallelism (number of processors used) and elapsed and CPU times used during the execution. To see a text-based actual plan run the command `SET STATISTICS PROFILE ON` before running the query. The query will execute, and you get the plan and the results.

Live Query Statistics

This plan viewing option combines the estimated and actual plans into an animated plan that displays execution progress through the operators in the plan. It refreshes every second and shows the actual number of rows flowing through the operators. The other benefit to Live Query Statistics is that it shows the handoff from operator to operator, which may be helpful in troubleshooting some performance issues. Because the type of plan is animated, it's only available as a graphical plan.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

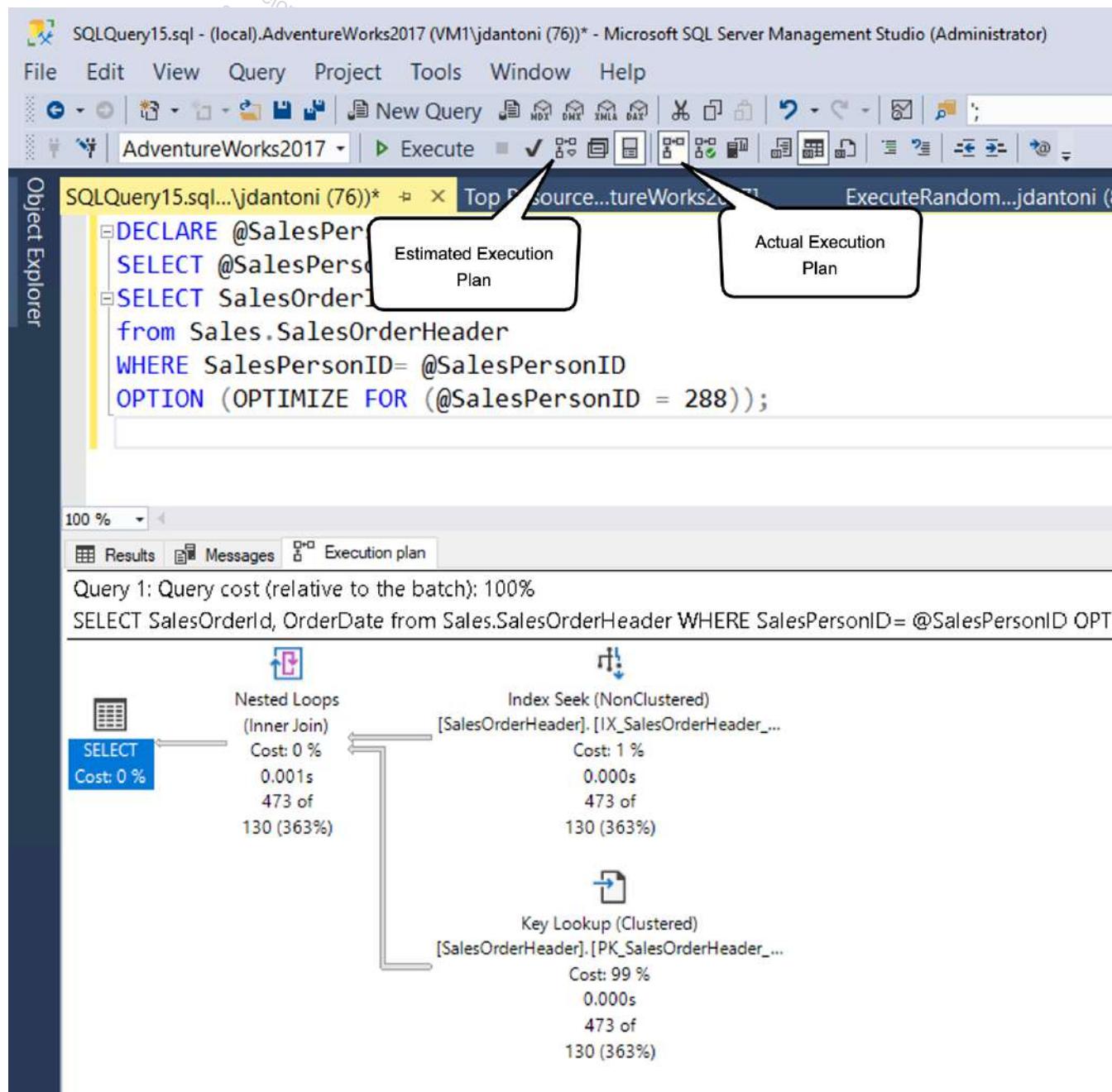
*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Explain estimated and actual query plans

The topic of actual versus estimated execution plans can be confusing. The difference is that the actual plan includes runtime statistics that aren't captured in the estimated plan. The operators used, and order of execution will be the same as the estimated plan in nearly all cases. The other consideration is that in order to capture an actual execution plan the query has to be executed, which can be time consuming, or not possible. For example, the query may be an `UPDATE` statement that can only be run once. However, if you need to see query results and the plan, you'll need to use one of the actual plan options.



As shown above, you can generate an estimated plan in SSMS by clicking the button pointed to by the estimated query plan box (or using the keyboard command **Control+L**). You can generate the actual plan by clicking the icon shown (or using the keyboard command **Control+M**), and then executing the query. The two option buttons work a bit differently. The *Include Estimated Query Plan* button responds immediately to any query highlighted (or the entire workspace, if nothing is highlighted), as opposed to *Include Actual Query Plan* button.

There's overhead to both executing a query and generating an estimated execution plan, so viewing execution plans should be done carefully in a production environment.

Usually you can use the estimated execution plan while writing your query, to understand its performance characteristics, identify missing indexes, or detect query anomalies. The actual execution plan is best used to understand the runtime performance of the query, and most importantly gaps in statistical data that cause the query optimizer to make suboptimal choices based on the data it has available.

Read a query plan

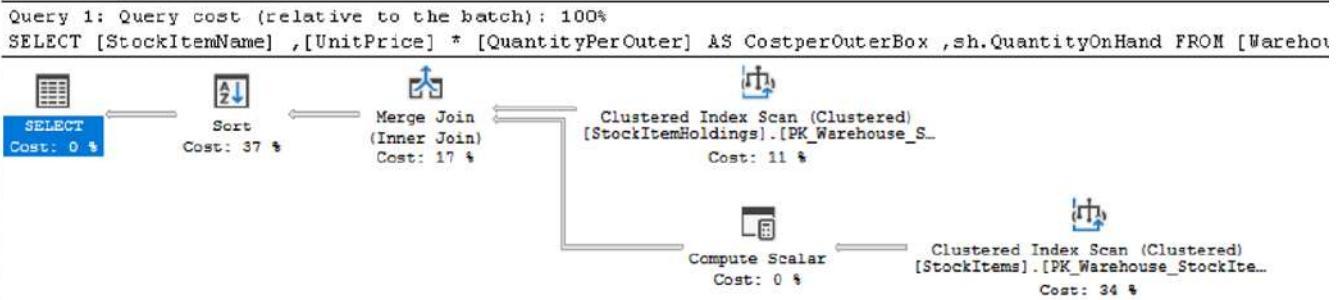
Execution plans show you what tasks the database engine is performing while retrieving the data needed to satisfy a query. Let's dive into the plan.

First, the query itself is shown below:

```
SELECT [stockItemName]
    ,[UnitPrice] * [QuantityPerOuter] AS CostPerOuterBox
    ,[QuantityOnHand]

FROM [Warehouse].[StockItems] s
JOIN [Warehouse].[StockItemHoldings] sh ON s.StockItemID = sh.StockItemID
ORDER BY CostPerOuterBox;
```

This query is joining the *StockItems* table to the *StockItemHoldings* table where the values in the column *StockItemID* are equal. The database engine has to first identify those rows before it can process the rest of the query.



Each icon in the plan shows a specific operation, which represents the various actions and decisions that make up an execution plan. The SQL Server database engine has over 100 query operators that can make up an execution plan. You'll notice that under each operator icon, there's a cost percentage relative to the total cost of the query. Even an operation that shows a cost of 0% still represents some cost. In fact, 0% is usually due to rounding, because the graphical plan costs are always shown as whole numbers, and the real percentage is something less than 0.5%.

The flow of execution in an execution plan is from right to left, and top to bottom, so in the plan above, the Clustered Index Scan operation on the *StockItemHoldings.PK_Warehouse_StockItemHoldings* clustered index is the first operation in the query. The widths of the lines that connect the operators are based on the estimated number of rows of data that flow onward to the next operator. A thick arrow is an indicator of large operator to operator transfer and may be indicative of an opportunity to tune a query. You can also hold your mouse over an operator and see additional information in a ToolTip as shown below.



```
== Clustered Index Scan (Clustered)
[StockItems].[PK_Warehouse_StockIte...
Cost: 34 %
```

Clustered Index Scan (Clustered)	
Scanning a clustered index, entirely or only a range.	
Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.0131613 (34%)
Estimated I/O Cost	0.0127546
Estimated Subtree Cost	0.0131613
Estimated CPU Cost	0.0004067
Estimated Number of Executions	1
Estimated Number of Rows	227
Estimated Number of Rows to be Read	227
Estimated Row Size	128 B
Ordered	True
Node ID	4
Object	
[WideWorldImporters].[Warehouse].[StockItems]. [PK_Warehouse_StockItems] [s]	
Output List	
[WideWorldImporters].[Warehouse].[StockItems].StockItemID, [WideWorldImporters].[Warehouse].[StockItems].StockItemName, [WideWorldImporters].[Warehouse]. [StockItems].QuantityPerOuter, [WideWorldImporters]. [Warehouse].[StockItems].UnitPrice	

The tooltip highlights the cost and estimates for the estimated plan, and for an actual plan will include the comparisons to the actual rows and costs. Each operator also has properties that will show you more than the tooltip does. If you right-click on a specific operator, you can select the Properties option from the context menu to see the full property list. This option will open up a separate Properties pane in SQL Server Management Studio, which by default is on the right side. Once the Properties pane is open, clicking on any operator will populate the Properties list with properties for that operator. Alternatively, you can open the Properties pane by clicking on View in the main SQL Server Management Studio menu and choosing Properties.

This document belongs to
Srikanth Srinivasan
sriv@skillpipe.com

Properties	
Clustered Index Scan (Clustered)	
Misc	
Defined Values	[WideWorldImporters].[Warehouse].[StockItems].St...
Description	Scanning a clustered index, entirely or only a range.
Estimated CPU Cost	0.0004067
Estimated Execution Mode	Row
Estimated I/O Cost	0.0127546
Estimated Number of Executions	1
Estimated Number of Rows	227
Estimated Number of Rows to be Read	227
Estimated Operator Cost	0.0131613 (34%)
Estimated Rebinds	0
Estimated Rewinds	0
Estimated Row Size	128 B
Estimated Subtree Cost	0.0131613
Forced Index	False
ForceScan	False
ForceSeek	False
Logical Operation	Clustered Index Scan
Node ID	4
NoExpandHint	False
Object	[WideWorldImporters].[Warehouse].[StockItems].[PK_V...
Alias	[s]
Database	[WideWorldImporters]
Index	[PK_Warehouse_StockItems]
Index Kind	Clustered
Schema	[Warehouse]
Storage	RowStore
Table	[StockItems]
Ordered	True
Output List	[WideWorldImporters].[Warehouse].[StockItems].Stock...
[1]	[WideWorldImporters].[Warehouse].[StockItems].Stock...
[2]	[WideWorldImporters].[Warehouse].[StockItems].Stock...
[3]	[WideWorldImporters].[Warehouse].[StockItems].Stock...
[4]	[WideWorldImporters].[Warehouse].[StockItems].Quan...
Parallel	False
Physical Operation	Clustered Index Scan
Scan Direction	FORWARD
Storage	RowStore
TableCardinality	227

The Properties pane includes some additional information and shows the output list, which provides details of the columns being passed to the next operator. These columns may indicate that a nonclustered index is needed to improve query performance when analyzed with clustered index scan. Since a clustered index scan operation is reading the entire table, in this scenario a non-clustered index on the *StockItemID* column in each table could be more efficient.

Lightweight query profiling

As mentioned above, capturing actual execution plans, whether using SSMS or the Extended Events monitoring infrastructure can have a large amount of overhead, and is typically only done in live site troubleshooting efforts. Observer overhead, as it's known, is the cost of monitoring a running application. In some scenarios, this cost can be just a few percentage points of CPU utilization, but in other cases like capturing actual execution plans, it can slow down individual query performance significantly. The legacy profiling infrastructure in SQL Server's engine could produce up to 75% overhead for capturing query information, whereas the lightweight profiling infrastructure has a maximum overhead of around 2%.

In the first version of lightweight profiling, it collected row count and I/O utilization information (the number of logical and physical reads and writes performed by the database engine to satisfy a given query). In addition, a new extended event called `query_thread_profile` was introduced to allow data from each operator in a query plan to be inspected. In the initial version of lightweight profiling, using the feature requires trace flag 7412 to be enabled globally.

In newer releases (SQL Server 2016 SP2 CU3, SQL Server 2017 CU11, and SQL Server 2019), if lightweight profiling isn't enabled globally, you can use the `USE HINT` query hint with `QUERY_PLAN_PROFILE` to enable lightweight profiling at the query level. When a query that has this hint completes execution, a `query_plan_profile` extended event is generated, which provides an actual execution plan. You can see an example of a query with this hint:

```
SELECT [stockItemName]
    ,[UnitPrice] * [QuantityPerOuter] AS CostPerOuterBox
    ,[QuantityOnHand]
FROM [Warehouse].[StockItems] s
    JOIN [Warehouse].[StockItems] sh ON s.StockItemID = sh.StockItemID
ORDER BY CostPerOuterBox
OPTION(USE HINT ('QUERY_PLAN_PROFILE'));
```

Last query plans stats

SQL Server 2019 and Azure SQL Database support two further enhancements to the query profiling infrastructure. First, lightweight profiling is enabled by default in both SQL Server 2019 and Azure SQL Database and managed instance. Lightweight profiling is also available as a database scoped configuration option, called `LIGHTWEIGHT_QUERY_PROFILING`. With the database scoped option, you can disable the feature for any of your user databases independent of each other.

Second, there's a new dynamic management function called `sys.dm_exec_query_plan_stats`, which can show you the last known actual query execution plan for a given plan handle. In order to see the last known actual query plan through the function, you can enable trace flag 2451 server-wide. Alternatively, you can enable this functionality using a database scoped configuration option called `LAST_QUERY_PLAN_STATS`.

You can combine this function with other objects to get the last execution plan for all cached queries as shown below:

```
SELECT *
FROM sys.dm_exec_cached_plans AS cp
    CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
    CROSS APPLY sys.dm_exec_query_plan_stats(plan_handle) AS qps;
GO
```

This functionality lets you quickly identify the runtime stats for the last execution of any query in your system, with minimal overhead. The image below shows how to retrieve the plan. If you select the execution plan XML, which will be the first column of results, it will display the execution plan shown in the second image below.

As you can see from the properties of the *Columnstore Index Scan* below, the plan retrieved from the cache has actual number of rows retrieved in the query.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

This document bel...

Describe dynamic management views and functions

SQL Server provides several hundred dynamic management objects. These objects contain system information that can be used to monitor the health of a server instance, diagnose problems, and tune performance. Dynamic management views and functions return internal data about the state of the database or the instance. Dynamic Management Objects can be either views (DMVs) or functions (DMFs), but most people use the acronym DMV to refer to both types of object.

There are two levels of DMVs, server scoped and database scoped.

- **Server scoped objects** – require `VIEW SERVER STATE` permission on the server
- **Database scoped objects** – require the `VIEW DATABASE STATE` permission within the database

The names of the DMVs are all prefixed with `sys.dmv_` followed by the functional area and then the specific function of the object. SQL Server supports three categories of DMVs:

- Database-related dynamic management objects
- Query execution related dynamic management objects
- Transaction related dynamic management objects

To learn about queries to monitor server and database performance, see [Monitoring Microsoft Azure SQL Database and Azure SQL Managed Instance performance using dynamic management views](#).

NOTE: For older versions of SQL Server where the query store is not available, you can use the view `sys.dm_exec_cached_plans` in conjunction with the functions `sys.dm_exec_sql_text` and `sys.dm_exec_query_plan` to return information about execution plans. However, unlike with Query Store, you will not be able to see changes in plans for a given query.

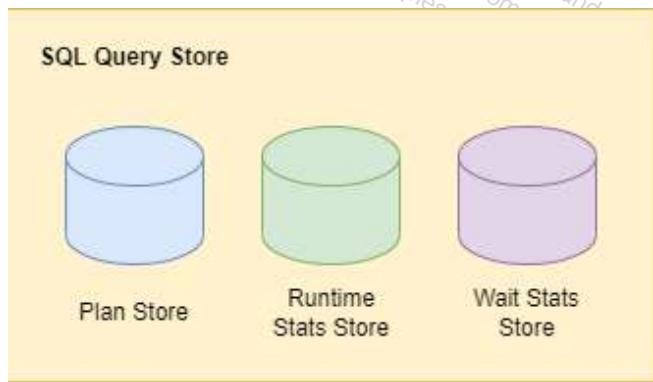
Azure SQL Database has a slightly different set of the DMVs available than SQL Server; some objects are available only in Azure SQL Database, while other objects are only available in SQL Server. Some are scoped at the server level and aren't applicable in the Azure model (the `waits_stats` DMV below is an example of a server-scoped DMV), while others are specific to Azure SQL Database, like `sys.dm_db_resource_stats` and provide Azure-specific information that isn't available in (or relevant to) SQL Server.

Explore Query Store

The SQL Server Query Store is a per-database feature that automatically captures a history of queries, plans, and runtime statistics to simplify performance troubleshooting and query tuning. It also provides insight into database usage patterns and resource consumption.

In total, the Query Store contains three stores:

- Plan store - used for storing estimated execution plan information
- Runtime stats store - used for storing execution statistics information
- Wait stats store - for persisting wait statistics information



Enable the Query Store

The Query Store is enabled by default in Azure SQL databases. If you want to use it with SQL Server and Azure Synapse Analytics, you need to enable it first. To enable the Query Store feature, use the following query valid for your environment:

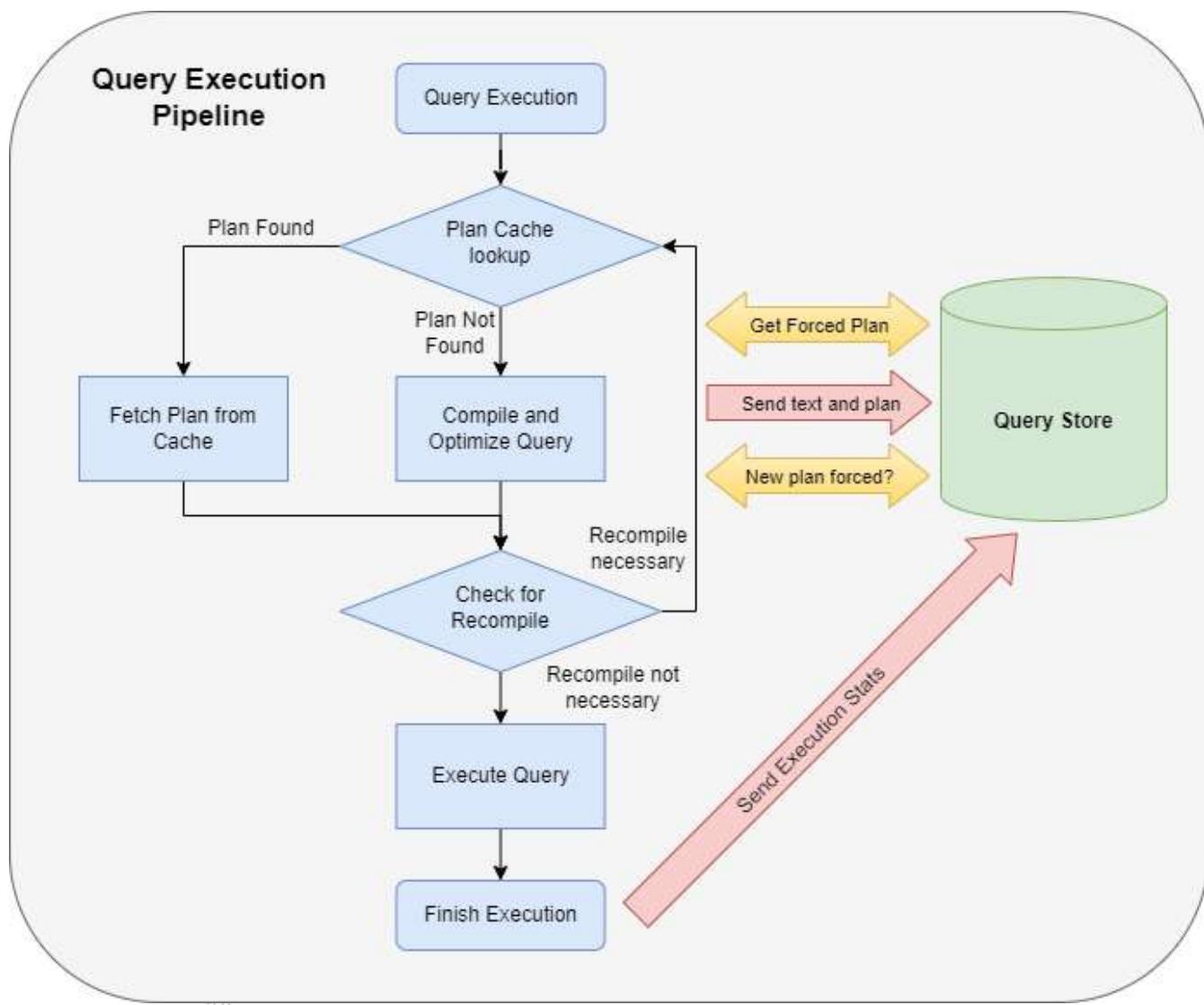
```
-- SQL Server  
ALTER DATABASE <database_name> SET QUERY_STORE = ON (OPERATION_MODE = READ_WRITE);  
  
-- Azure Synapse Analytics  
ALTER DATABASE <database_name> SET QUERY_STORE = ON;
```

How the Query Store collects data

The Query Store integrates with the query processing pipeline at many stages. Within each integration point, data is collected in memory and written to disk asynchronously to minimize I/O overhead. The integration points are as follows:

1. When a query executes for the first time, its query text and initial estimated execution plan are sent to the Query Store and persisted.
2. The plan updates in the Query Store when a query recompiles. If the recompile results in a newly generated execution plan, it also persists in the Query Store to augment the previous plans. In addition, the Query Store keeps track of the execution statistics for each query plan for comparison purposes.

3. During the compile and check for recompile phases, the Query Store identifies if there's a forced plan for the query to be executed. The query is recompiled if the Query Store provides a forced plan different from the plan in the procedure cache.
4. When a query executes, its runtime statistics persist in the Query Store. The Query Store aggregates this data to ensure an accurate representation of every query plan.



To learn more about how Query Store collects data, see [How Query Store collects data](#).

Common scenarios

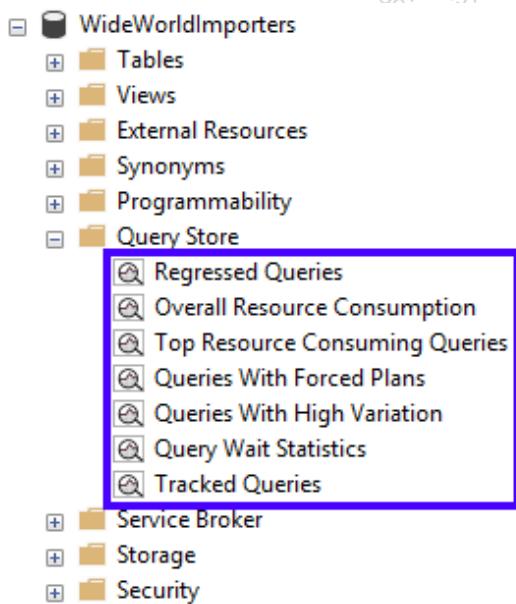
The SQL Server Query Store provides valuable insight into the performance of the operations performed in a database. The most common scenarios include:

- Identifying and fixing performance regression due to inferior query execution plan selection
- Identifying and tuning the highest resource consumption queries
- A/B testing to evaluate the impacts of database and application changes
- Ensuring performance stability after SQL Server upgrades
- Determining the most frequently used queries

- Audit the history of query plans for a query
- Identifying and improving ad hoc workloads
- Understand the prevalent wait categories of a database and the contributing queries and plans affecting wait times
- Analyze database usage patterns over time as it applies to resource consumption (CPU, I/O, Memory)

Discover the Query Store views

Once Query Store is enabled on a database, the Query Store folder is visible for the database in Object Explorer. For Azure Synapse Analytics, the Query Store views are displayed under System Views. The Query Store views provide aggregated, quick insights into the performance aspects of the SQL Server database.



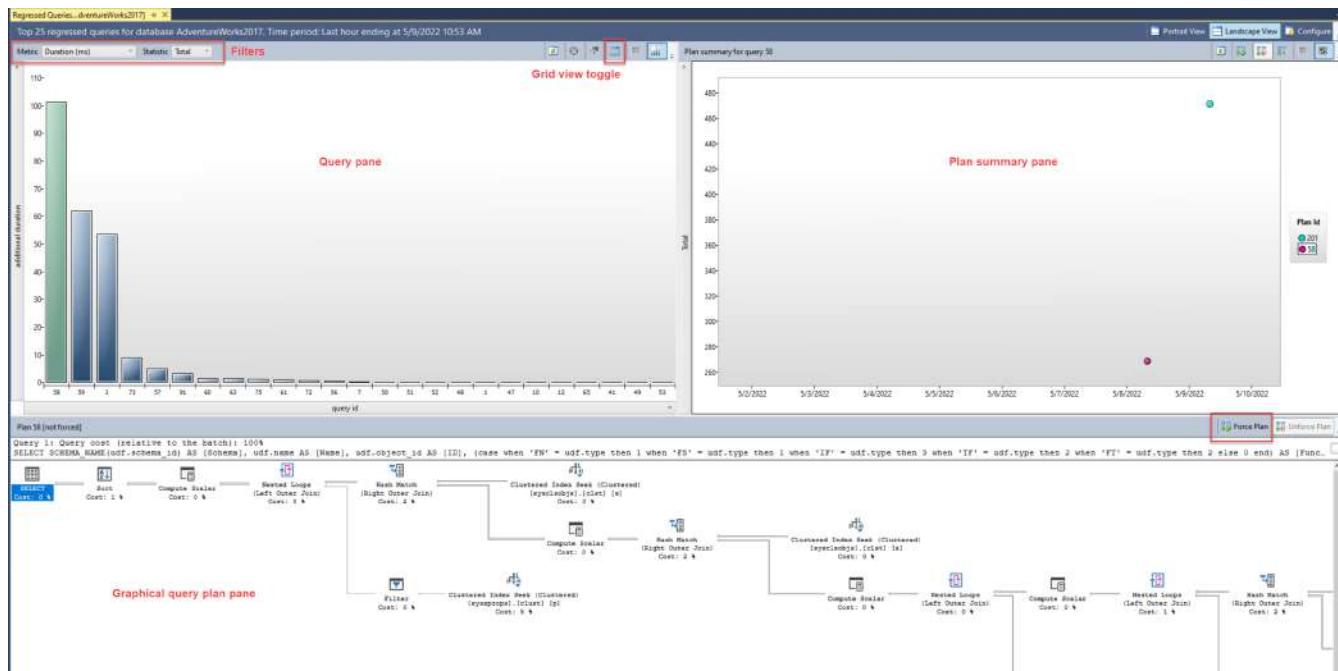
Regressed Queries

A regressed query is a query that is experiencing performance degradation over time due to execution plan changes. Estimated execution plans change due to many factors, including schema changes, statistics changes, and index changes. The first instinct may be to investigate the procedure cache, but the problem with the procedure cache is that it only stores the latest execution plan for a query; even then, plans are evicted based on the memory demands of the system. However, the Query Store persists several execution plans stored for each query, thus providing the flexibility to choose a specific plan in a concept known as *plan forcing* to solve the issue of a query performance regression caused by a plan change.

The **Regressed Queries** view can pinpoint queries whose execution metrics are regressing due to execution plan changes over a specified timeframe. The Regressed Queries view allows filtering based on selecting a metric (such as duration, CPU time, row count, and more) and a statistic (total, average, min, max, or standard deviation). Then, the view lists the top 25 regressed queries based on the filter provided. A graphical bar chart view of the queries displays by default, but you can optionally view the queries in a grid format.

The plan summary pane displays the persisted query plans associated with the query over time after selecting a query from the top-left query pane. You'll see a graphical query plan in the bottom pane by selecting a query plan in the Plan Summary pane. In addition, toolbar buttons are available in both the plan summary pane and graphical

query plan pane to force the selected plan for the selected query. This pane structure and behavior is consistently used across all SQL Query views.



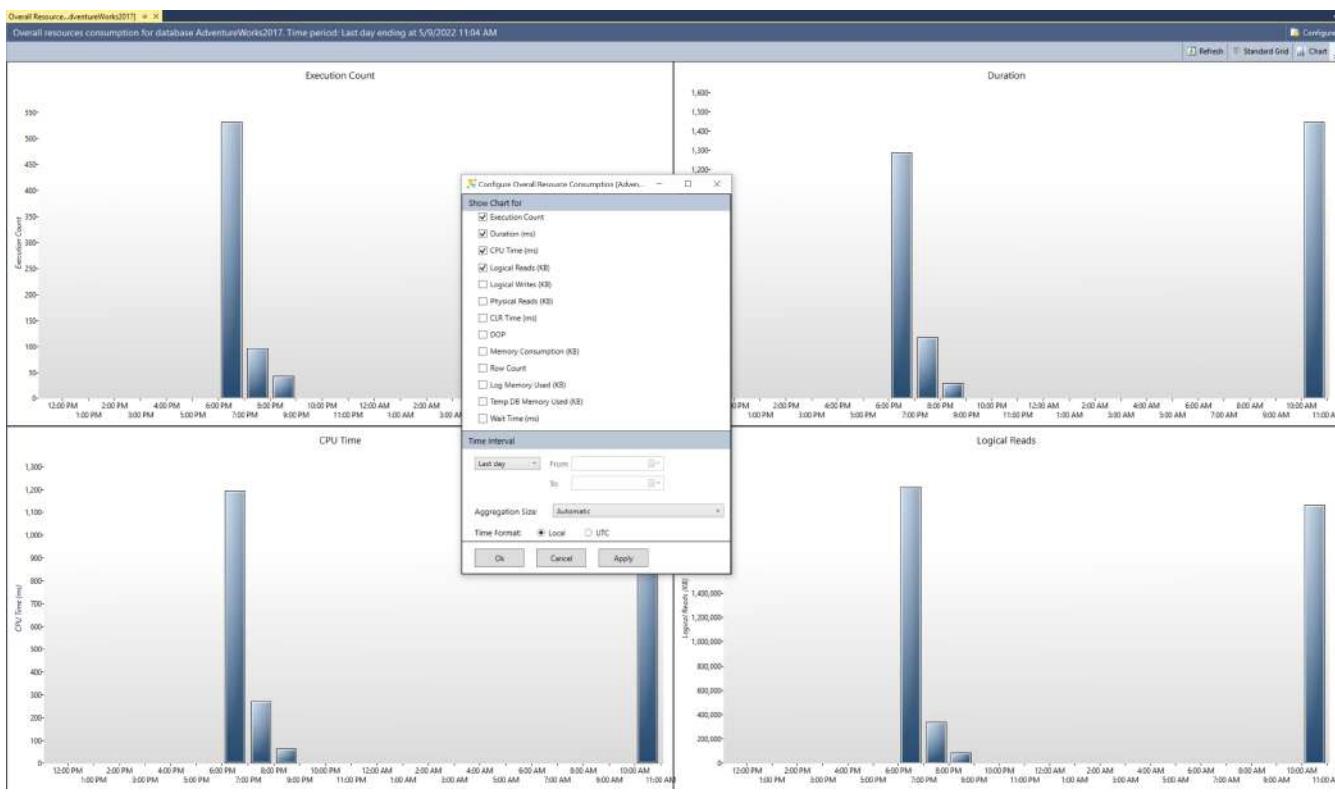
Alternatively, you can use the `sp_query_store_force_plan` stored procedure to use plan forcing.

```
EXEC sp_query_store_force_plan @query_id=73, @plan_id=79
```

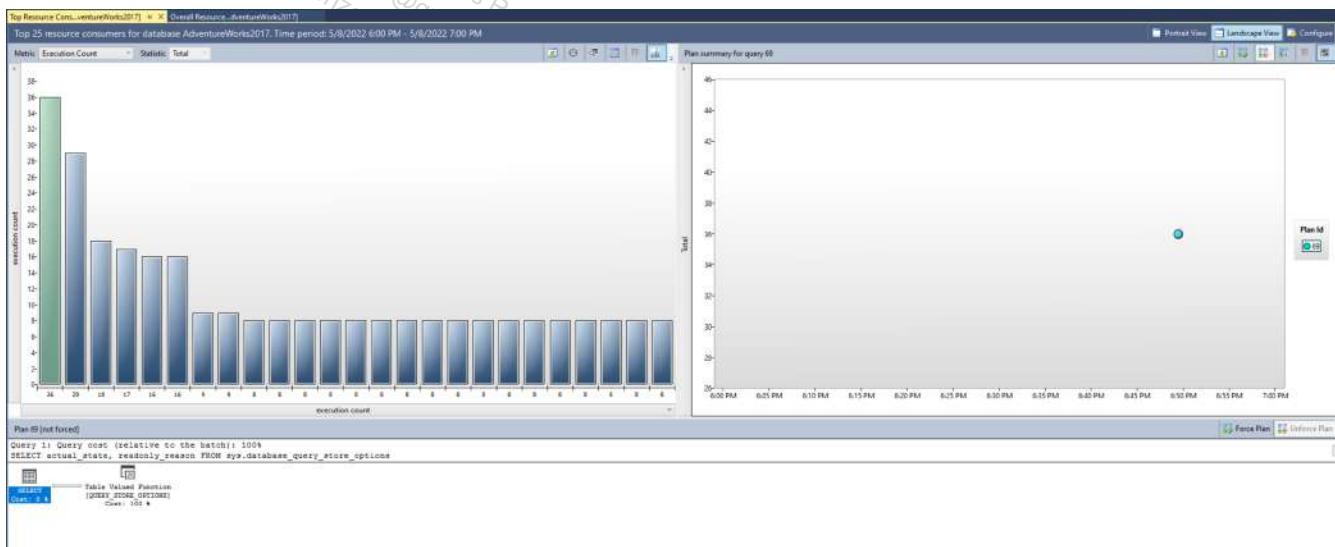
Overall Resource Consumption

The **Overall Resource Consumption** view allows for analyzing total resource consumption for multiple execution metrics (such as execution count, duration, wait time, and more) for a specified timeframe. The rendered charts are interactive; when selecting a measure from one of the charts, a drill through view displaying the queries associated with the chosen measure displays in a new tab.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

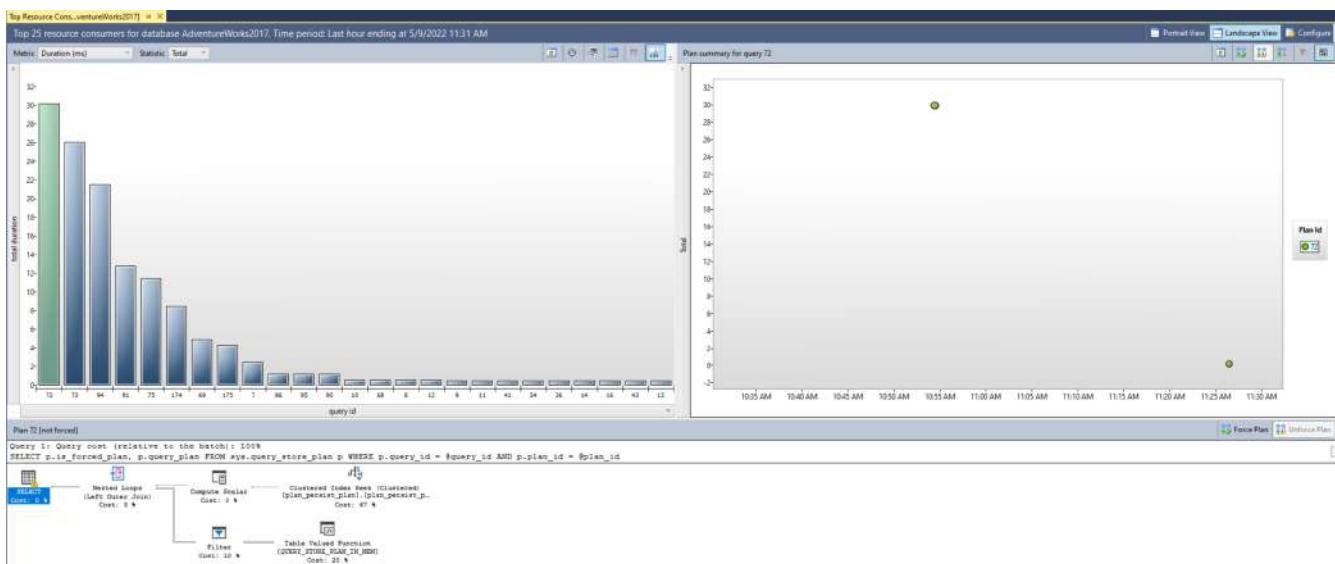


The details view provides the top 25 resource consumer queries that contributed to the metric that was selected. This details view uses the consistent interface that allows for the inspection of the associated queries and their details, evaluate saved estimated query plans, and optionally use plan forcing to improve performance. This view is valuable when system resource contention becomes an issue, such as when CPU usage reaches capacity.



Top Resource Consuming Queries

The **Top Resource Consuming Queries** view is similar to the details drill down of the Overall Resource Consumption view. It also allows for selecting a metric and a statistic as a filter. However, the queries it displays are the top 25 most impactful queries based on the chosen filter and timeframe.



The Top Resource Consuming Queries view provides the first indication of the ad hoc nature of the workload when identifying and improving ad hoc workloads. For example, in the following image, the *Execution Count* metric and the *Total* statistic are selected to unveil that approximately 90% of the top resource-consuming queries are only executed once.

Queries With Forced Plans

The **Queries With Forced Plans** view provides a quick look into the queries that have forced query plans. This view becomes relevant if a forced plan no longer performs as expected and needs to be reevaluated. This view provides the ability to review all persisted estimated execution plans for a selected query easily determining if another plan is now better suited for performance. If so, toolbar buttons are available to unforce a plan as required.

Queries With High Variation

Query performance can vary between executions. The **Queries with High Variation** view contains an analysis of queries that have the highest variation or standard deviation for a selected metric. The interface is consistent with most Query Store views allowing for query detail inspection, execution plan evaluation, and optionally forcing a specific plan. Use this view to tune unpredictable queries into a more consistent performance pattern.

Query Wait Statistics

The **Query Wait Statistics** view analyzes the most active wait categories for the database and renders a chart. This chart is interactive; selecting a wait category drills into the details of the queries that contribute to the wait time statistic.

The details view interface is also consistent with most query store views allowing for query detail inspection, execution plan evaluation, and optionally forcing a specific plan. This view helps identify queries that are affecting user experience across applications.

Tracking Query

The **Tracking Query** view allows analyzing a specific query based on an entered query ID value. Once run, the view provides the complete execution history of the query. A checkmark on an execution indicates a forced plan was used. This view can provide insight into queries such as those with forced plans to verify that query performance is remaining stable.

Using the Query Store to find query waits

When the performance of a system begins to degrade, it makes sense to consult query wait statistics to potentially identify a cause. In addition to identifying queries that need tuning, it can also shed light on potential infrastructure upgrades that would be beneficial.

The SQL Query Store provides the **Query Wait Statistics** view to provide insight into the top wait categories for the database. Currently, there are **23 wait categories**.

A bar chart displays the most impactful wait categories for the database when you open the Query Wait Statistics view. In addition, a filter located in the toolbar of the wait categories pane allows for the wait statistics to be calculated based on total wait time (default), average wait time, minimum wait time, maximum wait time, or standard deviation wait time.

Selecting a wait category will drill through to the details of the queries that contribute to that wait category. From this view, you have the ability to investigate individual queries that are the most impactful. You can access the persisted estimated execution plans display in the Plan summary pane by selecting a query in the query pane. Selecting a query plan from the Plan summary pane will display the graphical query plan in the bottom pane. From this view, you have the ability to force or unforce a query plan for the query to improve performance.

Identify problematic query plans

The path most DBAs take to troubleshoot query performance is to first identify the problematic query (typically the query consuming the highest amount of system resources), and then retrieve that query's execution plan. There are two scenarios. One is that the query consistently performs poorly. Consistent poor performance can be caused by a few different problems, including hardware resource constraints (though this situation typically won't affect a single query running in isolation), a suboptimal query structure, database compatibility settings, missing indexes, or poor choice of plan by the query optimizer. The second scenario is that the query performs well for some executions, but not others. This problem can be caused by a few other factors, the most common being data skew in a parameterized query that has an efficient plan for some executions, and a poor one for other executions. The other common factors in inconsistent query performance are blocking, where a query is waiting on another query to complete in order to gain access to a table, or hardware contention.

Let's look at each of these potential problems in more detail.

Hardware constraints

Usually, hardware constraints won't manifest themselves with single query executions but will be evident when production load is applied and there's a limited number of CPU threads and a limited amount of memory to be shared among the queries. When you have CPU contention, it will usually be detectable by observing the performance monitor counter '% Processor Time', which measures the CPU usage of the server. Looking deeper into SQL Server, you may see `SOS_SCHEDULER_YIELD` and `CXPACKET` wait types when the server is under CPU pressure. However, in some cases with poor storage system performance, even single executions of a query that is otherwise optimized can be slow. Storage system performance is best tracked at the operating system level using the performance monitor counters 'Disk Seconds/Read' and 'Disk Seconds/Write' which measure how long an I/O operation takes to complete. SQL Server will write to its error log if it detects poor storage performance (if an I/O takes longer than 15 seconds to complete). If you look at wait statistics and see a high percentage of `PAGEIOLATCH_SH` waits in your SQL Server, you might have a storage system performance issue. Typically, hardware performance is examined at a high level, early in the performance troubleshooting process, because it's relatively easy to evaluate.

Most database performance issues can be attributed to suboptimal query patterns, but in many cases running inefficient queries will put undue pressure on your hardware. For example, missing indexes could lead to CPU, storage, and memory pressure by retrieving more data than is required to process the query. It's recommended that you address suboptimal queries and tune them, before addressing hardware issue. We'll start looking at query tuning next.

Suboptimal query constructs

Relational databases perform best when executing set-based operations. Set-based operations perform data manipulation (`INSERT`, `UPDATE`, `DELETE`, and `SELECT`) in sets, where work is done on a set of values and produces either a single value or a result set. The alternative to set-based operations is to perform row-based work, using a cursor or a while loop. This type of processing is known as row-based processing, and its cost increases linearly with the number of rows impacted. That linear scale is problematic as data volumes grow for an application.

While detecting suboptimal use of row-based operations with cursors or WHILE loops is important, there are other SQL Server anti-patterns that you should be able to recognize. Table-valued functions (TVF), particularly multi-statement table-valued functions, caused problematic execution plan patterns prior to SQL Server 2017. Many developers like to use multi-statement table valued functions because they can execute multiple queries within a

single function and aggregate the results into a single table. However, anyone writing T-SQL code needs to be aware of the possible performance penalties for using TVFs.

SQL Server has two types of table-valued functions, inline and multi-statement. If you use an inline TVF, the database engine treats it just like a view. Multi-statement TVFs are treated just like another table when processing a query. Because TVFs are dynamic and as such, SQL Server doesn't have statistics on them, it used a fixed row count when estimating the query plan cost. A fixed count can be fine, if the number of rows is small, however if the TVF returns thousands or millions of rows, the execution plan could be inefficient.

Another anti-pattern has been the use of scalar functions, which have similar estimation and execution problems. Microsoft has made significant performance improvement with the introduction of Intelligent Query Processing, under compatibility levels 140 and 150.

SARGability

The term SARGable in relational databases refers to a predicate (WHERE clause) in a specific format that can use an index to speed up execution of a query. Predicates in the correct format are called 'Search Arguments' or SARGs. In SQL Server, using a SARG means that the optimizer will evaluate using a nonclustered index on the column referenced in the SARG for a *SEEK* operation, instead of scanning the entire index (or the entire table) to retrieve a value.

The presence of a SARG doesn't guarantee the use of an index for a *SEEK*. The optimizer's costing algorithms could still determine that the index was too expensive. This could be the case if a SARG refers to a large percentage of rows in a table. The absence of a SARG does mean that the optimizer won't even evaluate a *SEEK* on a nonclustered index.

Some examples of expressions that aren't SARGs (sometimes said to be non-sargable) are those that include a `LIKE` clause with a wildcard at the beginning of the string to be matched, for example, `WHERE LastName LIKE '%SMITH%'`. Other predicates that aren't SARGs occur when using functions on a column, for example, `WHERE CONVERT(CHAR(10), CreateDate, 121) = '2020-03-22'`. These queries with non-sargable expressions are typically identified by examining execution plans for index or table scans, where seeks should otherwise be taking place.

SQLQuery2.sql - VM...VM1\jdantoni (88)* ExecutionPlan1.sqlplan*

```
SELECT AddressID, AddressLine1, AddressLine2, City
FROM Person.Address
WHERE LEFT (City,1) = 'M'
```

100 % ↻

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

```
SELECT AddressID, AddressLine1, AddressLine2, City FROM Person.Address WHERE LEFT (City,1) = 'M'
```

The execution plan diagram shows a single operator node labeled "Index Scan (NonClustered)" with a cost of 99%. It has two inputs: one from a "SELECT" node with a cost of 1% and another from a "Address" table node.

Index Scan (NonClustered)	
Scan a nonclustered index, entirely or only a range.	
Physical Operation	Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.148561 (99%)
Estimated I/O Cost	0.126829
Estimated Subtree Cost	0.148561
Estimated CPU Cost	0.0217324
Estimated Number of Executions	1
Estimated Number of Rows	1216
Estimated Number of Rows to be Read	19614
Estimated Row Size	156 B
Ordered	False
Node ID	1
Predicate	
substring([AdventureWorks2017].[Person].[Address].[City],(1), (1))='M'	
Object	
[AdventureWorks2017].[Person].[Address].[IX_Address_City]	
Output List	
[AdventureWorks2017].[Person].[Address].AddressID, [AdventureWorks2017].[Person].[Address].AddressLine1, [AdventureWorks2017].[Person].[Address].AddressLine2, [AdventureWorks2017].[Person].[Address].City	

There's an index on the *City* column that is being used in the `WHERE` clause of the query and while it's being used in this execution plan above, you can see the index is being scanned, which means the entire index is being read. The `LEFT` function in the predicate makes this expression non-SARGable. The optimizer won't evaluate using an index seek on the index on the *City* column.

This query could be written to use a predicate that is SARGable. The optimizer would then evaluate a *SEEK* on the index on the *City* column. An index seek operator, in this case, would read a much smaller set of rows, as shown below.

SQLQuery2.sql - VM...VM1\jdantoni (88)* ➔ X ExecutionPlan1.sql\plan*

```
SELECT AddressID, AddressLine1, AddressLine2, City
FROM Person.Address
WHERE City LIKE 'M%'
```

100 %

Messages Execution plan

Query 1: Query cost (relative to the batch): 100%

SELECT AddressID, AddressLine1, AddressLine2, City FROM Person.Address WHERE City LIKE 'M%'

Index Seek (NonClustered)
[Address].[IX_Address_City]
Cost: 100 %

Index Seek (NonClustered)
Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0.0120842 (100%)
Estimated I/O Cost	0.0105324
Estimated Subtree Cost	0.0120842
Estimated CPU Cost	0.0015518
Estimated Number of Executions	1
Estimated Number of Rows	1267.98
Estimated Number of Rows to be Read	1267.98
Estimated Row Size	169 B
Ordered	True
Node ID	0

Predicate
[AdventureWorks2017].[Person].[Address].[City] like N'M%'

Object
[AdventureWorks2017].[Person].[Address].[IX_Address_City]

Output List
[AdventureWorks2017].[Person].[Address].AddressID,
[AdventureWorks2017].[Person].[Address].AddressLine1,
[AdventureWorks2017].[Person].[Address].AddressLine2,
[AdventureWorks2017].[Person].[Address].City

Seek Predicates
Seek Keys[1]: Start: [AdventureWorks2017].[Person].[Address].City >= Scalar Operator(N'M'), End: [AdventureWorks2017].[Person].[Address].City < Scalar Operator(N'N')

Changing **LEFT** function into a **LIKE** results in an index seek.

NOTE: The LIKE keyword, in this instance, does not have a wildcard on the left, so it is looking for cities that begin with M. , if it was “two-sided” or started with a wildcard (“%M%” or “%M”) it would be non-SARGable. The seek operation is estimated to return 1267 rows, or approximately 15% of the estimate for the query with the non-SARGable predicate.

Some other database development anti-patterns are treating the database as a service rather than a data store. Using a database to convert data to JSON, manipulate strings, or perform complex calculations can lead to excessive CPU use and increased latency. Queries that try to retrieve all records and then perform computations in the database can lead to excessive IO and CPU usage. Ideally, you should use the database for data access operations and optimized database constructs like aggregation.

Missing indexes

The most common performance problems we see as database administrators are due to a lack of useful indexes causing the engine to read far more pages than necessary to return the results of a query. While indexes aren't free in terms of resources (adding more indexes to a table can affect write performance and consume space), the performance gains they offer can offset the extra resource costs many times over. Frequently execution plans with these performance issues can be identified by the query operator *Clustered Index Scan* or the combination of the *Nonclustered Index Seek* and *Key Lookup* (which is more indicative of missing columns in an existing index).

The database engine attempts to help with this problem by reporting on missing indexes in execution plans. The names and details of the recommended indexes are available through a dynamic management view called

`sys.dm_db_missing_index_details`. There are also other DMVs in SQL Server like

`sys.dm_db_index_usage_stats` and `sys.dm_db_index_operational_stats`, which highlight the utilization of existing indexes.

It may make sense to drop an index that isn't used by any queries in the database. The missing index DMVs and plan warnings should only be used as a starting point for tuning your queries. It's important to understand what your key queries are and build indexes to support those queries. Creating all missing indexes without evaluating indexes in the context of each other isn't recommended.

Missing and out-of-date statistics

You've learned about the importance of column and index statistics to the query optimizer. It's also important to understand conditions that can lead to out-of-date statistics, and how this problem can manifest itself in SQL Server. Azure SQL offerings default to having auto-update statistics set to ON. Prior to SQL Server 2016, the default behavior of auto-update statistics was to not update statistics until the number of modifications to columns in the index was equal to about 20% of the number of rows in a table. Because of this behavior, you could have data modifications that were significant enough to change query performance, but not update the statistics. Any plan that used the table with the changed data would be based on out-of-date statistics and would frequently be suboptimal.

Prior to SQL Server 2016, you had the option of using trace flag 2371, which changed the required number of modifications to be a dynamic value, so as your table grew in size, the percentage of row modifications that was required to trigger a statistics update got smaller. Newer versions of SQL Server, Azure SQL Database and Azure SQL Managed Instance support this behavior by default. There's also a dynamic management function called `sys.dm_db_stats_properties`, which shows you the last time statistics were updated and the number of modifications that have been made since the last update, which allows you to quickly identify statistics that might need to be manually updated.

Poor optimizer choices

While the query optimizer does a good job of optimizing most queries, there are some edge cases where the cost-based optimizer may make impactful decisions that aren't fully understood. There are many ways to address this including using query hints, trace flags, execution plan forcing, and other adjustments in order to reach a stable and optimal query plan. Microsoft has a support team that can help troubleshoot these scenarios.

In the below example from the *AdventureWorks2017* database, a query hint is being used to tell the database optimizer to always use a city name of Seattle. This hint won't guarantee the best execution plan for all city values, but it will be predictable. The value of 'Seattle' for `@city_name` will only be used during optimization. During execution, the actual supplied value ('Ascheim') will be used.

```
DECLARE @city_name nvarchar(30) = 'Ascheim',
        @postal_code nvarchar(15) = 86171;

SELECT *
FROM Person.Address
WHERE City = @city_name
      AND PostalCode = @postal_code
OPTION (OPTIMIZE FOR (@city_name = 'Seattle'));
```

As seen in the example, the query uses a hint (the `OPTION` clause) to tell the optimizer to use a specific variable value to build its execution plan.

Parameter sniffing

SQL Server caches query execution plans for future use. Since the execution plan retrieval process is based on the hash value of a query, the query text has to be identical for every execution of the query for the cached plan to be used. In order to support multiple values in the same query, many developers use parameters, passed in through stored procedures, as seen in the example below:

```
CREATE PROC GetAccountID (@Param INT)
AS

<other statements in procedure>

SELECT accountid FROM Customersales WHERE sales > @Param;

<other statements in procedure>

RETURN;

-- Call the procedure:

EXEC GetAccountID 42;
```

Queries can also be explicitly parameterized using the procedure `sp_executesql`. However, explicit parameterization of individual queries is usually done through the application with some form (depending on the API) of *PREPARE* and *EXECUTE*. When the database engine executes that query for the first time, it will optimize the query based on the initial value of the parameter, in this case, 42. This behavior, called parameter sniffing, allows for the overall workload of compiling queries to be reduced on the server. However, if there's data skew, query performance could vary widely.

For example, a table that had 10 million records, and 99% of those records have an ID of 1, and the other 1% are unique numbers, performance will be based on which ID was initially used to optimize the query. This wildly fluctuating performance is indicative of data skew and isn't an inherent problem with parameter sniffing. This behavior is a fairly common performance problem that you should be aware of. You should understand the options for alleviating the issue. There are a few ways to address this problem, but they each come with tradeoffs:

- Use the `RECOMPILE` hint in your query, or the `WITH RECOMPILE` execution option in your stored procedures. This hint will cause the query or procedure to be recompiled every time it's executed, which will increase CPU utilization on the server but will always use the current parameter value.
- You can use the `OPTIMIZE FOR UNKNOWN` query hint. This hint will cause the optimizer to choose to not sniff parameters and compare the value with column data histogram. This option won't get you the best possible plan but will allow for a consistent execution plan.
- Rewrite your procedure or queries by adding logic around parameter values to only RECOMPILE for known troublesome parameters. In the example below, if the SalesPersonID parameter is NULL, the query will be executed with the `OPTION (RECOMPILE)`.

```
CREATE OR ALTER PROCEDURE GetsalesInfo (@SalesPersonID INT = NULL)
AS
DECLARE @Recompile BIT = 0
      , @SQLString NVARCHAR(500)

SELECT @SQLString = N'SELECT SalesorderId, orderDate FROM Sales.SalesOrderHeader WHERE
SalesPersonID = @SalesPersonID'

IF @SalesPersonID IS NULL
BEGIN
    SET @Recompile = 1
END

IF @Recompile = 1
BEGIN
    SET @SQLString = @SQLString + N' OPTION(RECOMPILE)'
END

EXEC sp_executesql @SQLString
    ,N'@SalesPersonID INT'
    ,@SalesPersonID = @SalesPersonID
GO
```

The example above is a good solution but it does require a fairly large development effort, and a firm understanding of your data distribution. It also may require maintenance as the data changes.

Describe blocking and locking

One feature of relational databases is locking, which is essential to maintain the atomicity, consistency, and isolation properties of the ACID model. All RDBMSs will block actions that would violate the consistency and isolation of writes to a database. SQL programmers are responsible for starting and ending transactions at the right point, in order to ensure the logical consistency of their data. In turn, the database engine provides locking mechanisms that also protect the logical consistency of the tables affected by those queries. These actions are a foundational part of the relational model.

On SQL Server, blocking occurs when one process holds a lock on a specific resource (row, page, table, database), and a second process attempts to acquire a lock with an incompatible lock type on the same resource. Typically, locks are held for a short period, and when the process holding the lock releases it, the blocked process can then acquire the lock and complete its transaction.

SQL Server locks the smallest amount of data needed to successfully complete the transaction. This behavior allows maximum concurrency. For example, if SQL Server is locking a single row, all other rows in the table are available for other processes to use, so concurrent work can go on. However, each lock requires memory resources, so it's not cost-effective for one process to have thousands of individual locks on a single table. SQL Server tries to balance concurrency with cost. One technique used is called lock escalation. If SQL Server needs to lock more than 5000 rows on a single object in a single statement, it will escalate the multiple row locks to a single table lock.

Locking is normal behavior and happens many times during a normal day. Locking only becomes a problem when it causes blocking that isn't quickly resolved. There are two types of performance issues that can be caused by blocking:

- A process holds locks on a set of resources for an extended period of time before releasing them. These locks cause other processes to block, which can degrade query performance and concurrency.
- A process gets locks on a set of resources, and never releases them. This problem requires administrator intervention to resolve.

Another blocking scenario is deadlocking, which occurs when one transaction has a lock on a resource, and another transaction has a lock on a second resource. Each transaction then attempts to take a lock on the resource, which is currently locked by the other transaction. Theoretically, this scenario would lead to an infinite wait, as neither transaction could complete. However, the SQL Server engine has a mechanism for detecting these scenarios and will kill one of the transactions in order to alleviate the deadlock, based on which transaction has performed the least amount of work that would need to be rolled back. The transaction that is killed is known as the deadlock victim. Deadlocks are recorded in the `system_health` extended event session, which is enabled by default.

It's important to understand the concept of a transaction. Auto-commit is the default mode of SQL Server and Azure SQL Database, which means the changes made by the statement below would automatically be recorded in the database's transaction log.

```
INSERT INTO DemoTable (A) VALUES (1);
```

In order to allow developers to have more granular control over their application code, SQL Server also allows you to explicitly control your transactions. The query below would take a lock on a row in the `DemoTable` table what wouldn't be released until a subsequent command to commit the transaction was added.

```
BEGIN TRANSACTION
INSERT INTO DemoTable (A) VALUES (1);
```

The proper way to write the above query is as follows:

```
BEGIN TRANSACTION
INSERT INTO DemoTable (A) VALUES (1);
COMMIT TRANSACTION
```

The `COMMIT TRANSACTION` command explicitly commits a record of the changes to the transaction log. The changed data will eventually make its way into the data file asynchronously. These transactions represent a unit of work to the database engine. If the developer forgets to issue the `COMMIT TRANSACTION` command, the transaction will stay open and the locks won't be released. This is one of the main reasons for long running transactions.

The other mechanism the database engine uses to help the concurrency of the database is row versioning. When a row versioning isolation level is enabled to the database, engine maintains versions of each modified row in TempDB. This is typically used in mixed use workloads, in order to prevent reading queries from blocking queries that are writing to the database.

To monitor open transactions awaiting commit or rollback run the following query:

```
SELECT tst.session_id, [database_name] = db_name(s.database_id)
    , tat.transaction_begin_time
    , transaction_duration_s = datediff(s, tat.transaction_begin_time, sysdatetime())
    , transaction_type = CASE tat.transaction_type WHEN 1 THEN 'Read/write transaction'
                                WHEN 2 THEN 'Read-only transaction'
                                WHEN 3 THEN 'System transaction'
                                WHEN 4 THEN 'Distributed transaction' END
    , input_buffer = ib.event_info, tat.transaction_uow
    , transaction_state = CASE tat.transaction_state
                                WHEN 0 THEN 'The transaction has not been completely initialized yet.'
                                WHEN 1 THEN 'The transaction has been initialized but has not started.'
                                WHEN 2 THEN 'The transaction is active - has not been committed or rolled back.'
                                WHEN 3 THEN 'The transaction has ended. This is used for read-only transactions.'
                                WHEN 4 THEN 'The commit process has been initiated on the distributed transaction.'
                                WHEN 5 THEN 'The transaction is in a prepared state and waiting resolution.'
                                WHEN 6 THEN 'The transaction has been committed.'
                                WHEN 7 THEN 'The transaction is being rolled back.'
                                WHEN 8 THEN 'The transaction has been rolled back.' END
    , transaction_name = tat.name, request_status = r.status
    , tst.is_user_transaction, tst.is_local
    , session_open_transaction_count = tst.open_transaction_count
    , s.host_name, s.program_name, s.client_interface_name, s.login_name, s.is_user_process
FROM sys.dm_tran_active_transactions tat
INNER JOIN sys.dm_tran_session_transactions tst ON tat.transaction_id = tst.transaction_id
INNER JOIN sys.dm_exec_sessions s ON s.session_id = tst.session_id
LEFT OUTER JOIN sys.dm_exec_requests r ON r.session_id = s.session_id
CROSS APPLY sys.dm_exec_input_buffer(s.session_id, null) AS ib
ORDER BY tat.transaction_begin_time DESC;
```

Isolation levels

SQL Server offers several isolation levels to allow you to define the level of consistency and correctness you need guaranteed for your data. Isolation levels let you find a balance between concurrency and consistency. The isolation level doesn't affect the locks taken to prevent data modification, a transaction will always get an exclusive lock on the data that is modifying. However, your isolation level can affect the length of time that your locks are held. Lower isolation levels increase the ability of multiple user process to access data at the same time, but increase the data consistency risks that can occur. The isolation levels in SQL Server are as follows:

- **Read uncommitted** – Lowest isolation level available. Dirty reads are allowed, which means one transaction may see changes made by another transaction that haven't yet been committed.
- **Read committed** – allows a transaction to read data previously read, but not modified by another transaction with or without waiting for the first transaction to finish. This level also releases read locks as soon as the select operation is performed. This is the default SQL Server level.
- **Repeatable Read** – This level keeps read and write locks that are acquired on selected data until the end of the transaction.
- **Serializable** – This is the highest level of isolation where transactions are isolated. Read and write locks are acquired on selected data and not released until the end of the transaction.

SQL Server also includes two isolation levels that include row-versioning.

- **Read Committed Snapshot** – In this level read operations take no row or page locks, and the engine presents each operation with a consistent snapshot of the data as it existed at the start of the query. This level is typically used when users are running frequent reporting queries against an OLTP database, in order to prevent the read operations from blocking the write operations.
- **Snapshot** – This level provides transaction level read consistency through row versioning. This level is vulnerable to update conflicts. If a transaction running under this level reads data modified by another transaction, an update by the snapshot transaction will be terminated and roll back. This isn't an issue with read committed snapshot isolation.

Isolation levels are set for each session with the T-SQL `SET` command, as shown:

```
SET TRANSACTION ISOLATION LEVEL  
{ READ UNCOMMITTED  
| READ COMMITTED  
| REPEATABLE READ  
| SNAPSHOT  
| SERIALIZABLE  
}
```

There's no way to set a global isolation level all queries running in a database, or for all queries run by a particular user. It's a session level setting.

Monitoring for blocking problems

Identifying blocking problem can be troublesome as they can be sporadic in nature. There's a DMV called `sys.dm_tran_locks`, which can be joined with `sys.dm_exec_requests` in order to provide further information on locks that each session is holding. A better way to monitor for blocking problems is to do so on an ongoing basis using the Extended Events engine.

Blocking problems typically fall into two categories:

- Poor transactional design. As shown above, a transaction that has no `COMMIT TRANSACTION` will never end. While that is a simple example, trying to do too much work in a single transaction or having a distributed transaction, which uses a linked server connection, can lead to unpredictable performance.
- Long running transactions caused by schema design. Frequently this can be an update on a column with a missing index, or poorly designed update query.

Monitoring for locking-related performance problems allows you to quickly identify performance degradation related to locking.

For more information about how to monitor blocking, see [Understand and resolve SQL Server blocking problems](#).

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

Which type of execution plan is stored in the plan cache?

- Estimated execution plan
- Actual execution plan
- Live Query Stats

Check Answers

Multiple choice

Which DMV should you use to find index utilization?

- sys.dm_db_index_usage_stats
- sys.dm_db_missing_index_details
- sys.dm_exec_query_plan_stats

Check Answers

Multiple choice

Which of the following wait types would indicate excessive CPU consumption?

- SOS_SCHEDULER_YIELD
- RESOURCE_SEMAPHORE
- PAGEIOLATCH_SH

Check Answers

Multiple choice

Which isolation level should you choose if you want to prevent users reading data from blocking users writing data?

- Serializable
- Read Committed Snapshot Isolation
- Repeatable Read

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

Database performance is determined by how execution plans work and how indexes are used. At the same time, you need to understand patterns and anti-patterns for relational databases and aim for good query choices and proper indexing. Query Store acts as a data recorder for the database engine, and it is focused on query performance and changes in execution plans.

Now that you've reviewed this module, you should be able to:

- Compare the different types of execution plans
- Understand the purpose and benefits of the Query Store
- Investigate the available reports and data in the Query Store
- Understand how blocking and locking work in the SQL Server database engine

Introduction

Database design is an important aspect of database performance, even though it's not always under the control of the database administrator. You may be working with third-party vendor applications that you didn't build.

Whenever possible, it's important to design your database properly for the workload, whether it's an online transaction processing (OLTP) or data warehouse workload. Many design decisions, such as choosing the right datatypes, can make large differences in the performance of your databases.

Learning objectives:

In this module, you will:

- Understand normal form and how it affects database design
- Choose appropriate data types for your database
- Understand types of indexes

Describe normalization

Database normalization is a design process used to organize a given set of data into tables and columns in a database. Each table should contain data relating to a specific ‘thing’ and only have data that supports that same ‘thing’ included in the table. The goal of this process is to reduce duplicate data contained within your database, to reduce performance degradation of database inserts and updates. For example, a customer address change is much easier to implement if the only place of the customer address is stored in the Customers table. The most common forms of normalization are first, second, and third normal form and are described below.

First normal form

First normal form has the following specifications:

- Create a separate table for each set of related data
- Eliminate repeating groups in individual tables
- Identify each set of related data with a primary key

In this model, you shouldn't use multiple columns in a single table to store similar data. For example, if product can come in multiple colors, you shouldn't have multiple columns in a single row containing the different color values. The first table, below (ProductColors), isn't in first normal form as there are repeating values for color. For products with only one color, there's wasted space. And what if a product came in more than three colors? Rather than having to set a maximum number of colors, we can recreate the table as shown in the second table, ProductColor. We also have a requirement for first normal form that there's a unique key for the table, which is column (or columns) whose value uniquely identifies the row. Neither of the columns in the second table is unique, but together, the combination of ProductID and Color is unique. When multiple columns are needed, we call that a composite key.

ProductID	Color1	Color2	Color3
1	Red	Green	Yellow
2	Yellow		
3	Blue	Red	
4	Blue		
5	Red		

ProductID	Color
1	Red
1	Green
1	Yellow
2	Yellow

ProductID	Color
3	Blue
3	Red
4	Blue
5	Red

The third table, ProductInfo, is in first normal form because each row refers to a particular product, there are no repeating groups, and we have the column ProductID to use as a Primary Key.

ProductID	ProductName	Price	ProductionCountry	ShortLocation
1	Widget	15.95	United States	US
2	Foop	41.95	United Kingdom	UK
3	Glombit	49.95	United Kingdom	UK
4	Sorfin	99.99	Republic of the Philippines	RepPhil
5	Stem Bolt	29.95	United States	US

Second normal form

Second normal form has the following specification, in addition to those required by first normal form:

- If the table has a composite key, all attributes must depend on the complete key and not just part of it.

Second normal form is only relevant to tables with composite keys, like in the table ProductColor, which is the second table above. Consider the case where the ProductColor table also includes the product's price. This table has a composite key on ProductID and Color, because only using both column values can we uniquely identify a row. If a product's price doesn't change with the color, we might see data as shown in this table:

ProductID	Color	Price
1	Red	15.95
1	Green	15.95
1	Yellow	15.95
2	Yellow	41.95
3	Blue	49.95
3	Red	49.95
4	Blue	99.95

ProductID	Color	Price
5	Red	29.95

The table above is **not** in second normal form. The price value is dependent on the ProductID but not on the Color. There are three rows for ProductID 1, so the price for that product is repeated three times. The problem with violating second normal form is that if we have to update the price, we have to make sure we update it everywhere. If we update the price in the first row, but not the second or third, we would have something called an ‘update anomaly’. After the update, we wouldn’t be able to tell what the actual price for ProductID 1 was. The solution is to move the Price column to a table that has ProductID as a single column key, because that is the only column that Price depends on. For example, we could use Table 3 to store the Price.

If the price for a product was different based on its color, the fourth table would be in the second normal form, since the price would depend on both parts of the key: the ProductID and the Color.

Third normal form

Third normal form is typically the aim for most OLTP databases. Third normal form has the following specification, in addition to those required by second normal form:

- All nonkey columns are non-transitively dependent on the primary key.

The transitive relationship implies that one column in a table is related to other columns, through a second column. Dependency means that a column can derive its value from another, as a result of a dependency. For example, your age can be determined from your date of birth, making your age dependent on your date of birth. Refer back to the third table, ProductInfo. This table is in second normal form, but not in third. The *ShortLocation* column is dependent on the *ProductionCountry* column, which isn't the key. Like second normal form, violating third normal form can lead to update anomalies. We would end up with inconsistent data if we updated the *ShortLocation* in one row but didn't update it in all the rows where that location occurred. To prevent this, we could create a separate table to store country names and their shortened forms.

Denormalization

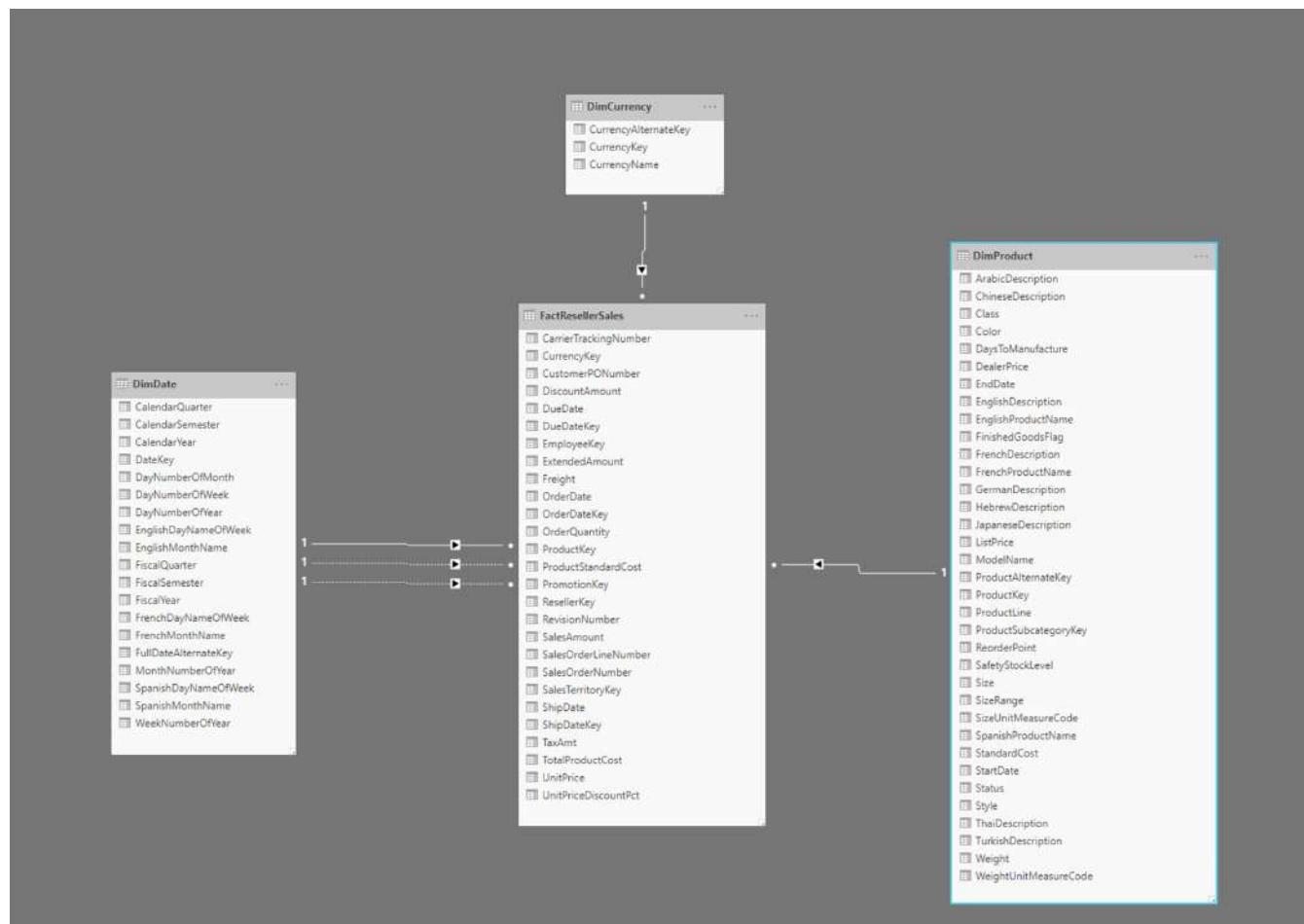
While the third normal form is theoretically desirable, it isn't always possible for all data. In addition, a normalized database doesn't always give you the best performance. Normalized data frequently requires multiple join operations to get all the necessary data returned in a single query. There's a tradeoff between normalizing data when the number of joins required to return query results has high CPU utilization, and denormalized data that has fewer joins and less CPU required, but opens up the possibility of update anomalies.

NOTE: Denormalized data is not the same as unnormalized. For denormalization, we start by designing tables that are normalized. Then we can add additional columns to some tables to reduce the number of joins required, but as we do so, we are aware of the possible update anomalies. We then make sure we have triggers or other kinds of processing that will make sure that when we perform an update, all the duplicate data is also updated.

Denormalized data can be more efficient to query, especially for read heavy workloads like a data warehouse. In those cases, having extra columns may offer better query patterns and/or more simplistic queries.

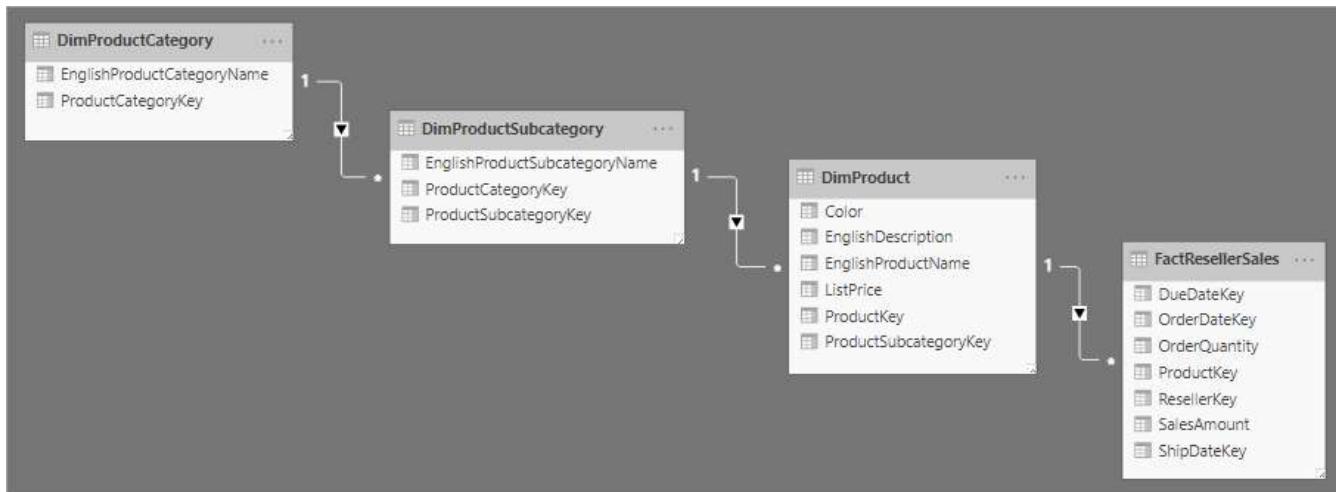
Star schema

While most normalization is aimed at OLTP workloads, data warehouses have their own modeling structure, which is usually a **denormalized** model. This design uses fact tables, which record measurements or metrics for specific events like a sale, and joins them to dimension tables, which are smaller in terms of row count, but may have a large number of columns to describe the fact data. Some example dimensions would include inventory, time, and/or geography. This design pattern is used to make the database easier to query and offer performance gains for read workloads.



The above image shows an example of a star schema, including a *FactResellerSales* fact table, and dimensions for date, currency, and products. The fact table contains data related to the sales transactions, and the dimensions only contain data related to a specific element of the sales data. For example, the *FactResellerSales* table contains only a *ProductKey* to indicate which product was sold. All of the details about each product are stored in the *DimProduct* table, and related back to the fact table with the *ProductKey* column.

Related to star schema design is a snowflake schema, which uses a set of more normalized tables for a single business entity. The following image shows an example of a single dimension for a snowflake schema. The Products dimension is normalized and stored in three tables called *DimProductCategory*, *DimProductSubcategory*, and *DimProduct*.



The main difference between star and snowflake schemas is that the dimensions in a snowflake schema are normalized to reduce redundancy, which saves storage space. The tradeoff is that your queries require more joins, which can increase your complexity and decrease performance.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Describe normalization

Database normalization is a design process used to organize a given set of data into tables and columns in a database. Each table should contain data relating to a specific ‘thing’ and only have data that supports that same ‘thing’ included in the table. The goal of this process is to reduce duplicate data contained within your database, to reduce performance degradation of database inserts and updates. For example, a customer address change is much easier to implement if the only place of the customer address is stored in the Customers table. The most common forms of normalization are first, second, and third normal form and are described below.

First normal form

First normal form has the following specifications:

- Create a separate table for each set of related data
- Eliminate repeating groups in individual tables
- Identify each set of related data with a primary key

In this model, you shouldn't use multiple columns in a single table to store similar data. For example, if product can come in multiple colors, you shouldn't have multiple columns in a single row containing the different color values. The first table, below (ProductColors), isn't in first normal form as there are repeating values for color. For products with only one color, there's wasted space. And what if a product came in more than three colors? Rather than having to set a maximum number of colors, we can recreate the table as shown in the second table, ProductColor. We also have a requirement for first normal form that there's a unique key for the table, which is column (or columns) whose value uniquely identifies the row. Neither of the columns in the second table is unique, but together, the combination of ProductID and Color is unique. When multiple columns are needed, we call that a composite key.

ProductID	Color1	Color2	Color3
1	Red	Green	Yellow
2	Yellow		
3	Blue	Red	
4	Blue		
5	Red		

ProductID	Color
1	Red
1	Green
1	Yellow
2	Yellow

ProductID	Color
3	Blue
3	Red
4	Blue
5	Red

The third table, ProductInfo, is in first normal form because each row refers to a particular product, there are no repeating groups, and we have the column ProductID to use as a Primary Key.

ProductID	ProductName	Price	ProductionCountry	ShortLocation
1	Widget	15.95	United States	US
2	Foop	41.95	United Kingdom	UK
3	Glombit	49.95	United Kingdom	UK
4	Sorfin	99.99	Republic of the Philippines	RepPhil
5	Stem Bolt	29.95	United States	US

Second normal form

Second normal form has the following specification, in addition to those required by first normal form:

- If the table has a composite key, all attributes must depend on the complete key and not just part of it.

Second normal form is only relevant to tables with composite keys, like in the table ProductColor, which is the second table above. Consider the case where the ProductColor table also includes the product's price. This table has a composite key on ProductID and Color, because only using both column values can we uniquely identify a row. If a product's price doesn't change with the color, we might see data as shown in this table:

ProductID	Color	Price
1	Red	15.95
1	Green	15.95
1	Yellow	15.95
2	Yellow	41.95
3	Blue	49.95
3	Red	49.95
4	Blue	99.95

ProductID	Color	Price
5	Red	29.95

The table above is **not** in second normal form. The price value is dependent on the ProductID but not on the Color. There are three rows for ProductID 1, so the price for that product is repeated three times. The problem with violating second normal form is that if we have to update the price, we have to make sure we update it everywhere. If we update the price in the first row, but not the second or third, we would have something called an ‘update anomaly’. After the update, we wouldn’t be able to tell what the actual price for ProductID 1 was. The solution is to move the Price column to a table that has ProductID as a single column key, because that is the only column that Price depends on. For example, we could use Table 3 to store the Price.

If the price for a product was different based on its color, the fourth table would be in the second normal form, since the price would depend on both parts of the key: the ProductID and the Color.

Third normal form

Third normal form is typically the aim for most OLTP databases. Third normal form has the following specification, in addition to those required by second normal form:

- All nonkey columns are non-transitively dependent on the primary key.

The transitive relationship implies that one column in a table is related to other columns, through a second column. Dependency means that a column can derive its value from another, as a result of a dependency. For example, your age can be determined from your date of birth, making your age dependent on your date of birth. Refer back to the third table, ProductInfo. This table is in second normal form, but not in third. The *ShortLocation* column is dependent on the *ProductionCountry* column, which isn't the key. Like second normal form, violating third normal form can lead to update anomalies. We would end up with inconsistent data if we updated the *ShortLocation* in one row but didn't update it in all the rows where that location occurred. To prevent this, we could create a separate table to store country names and their shortened forms.

Denormalization

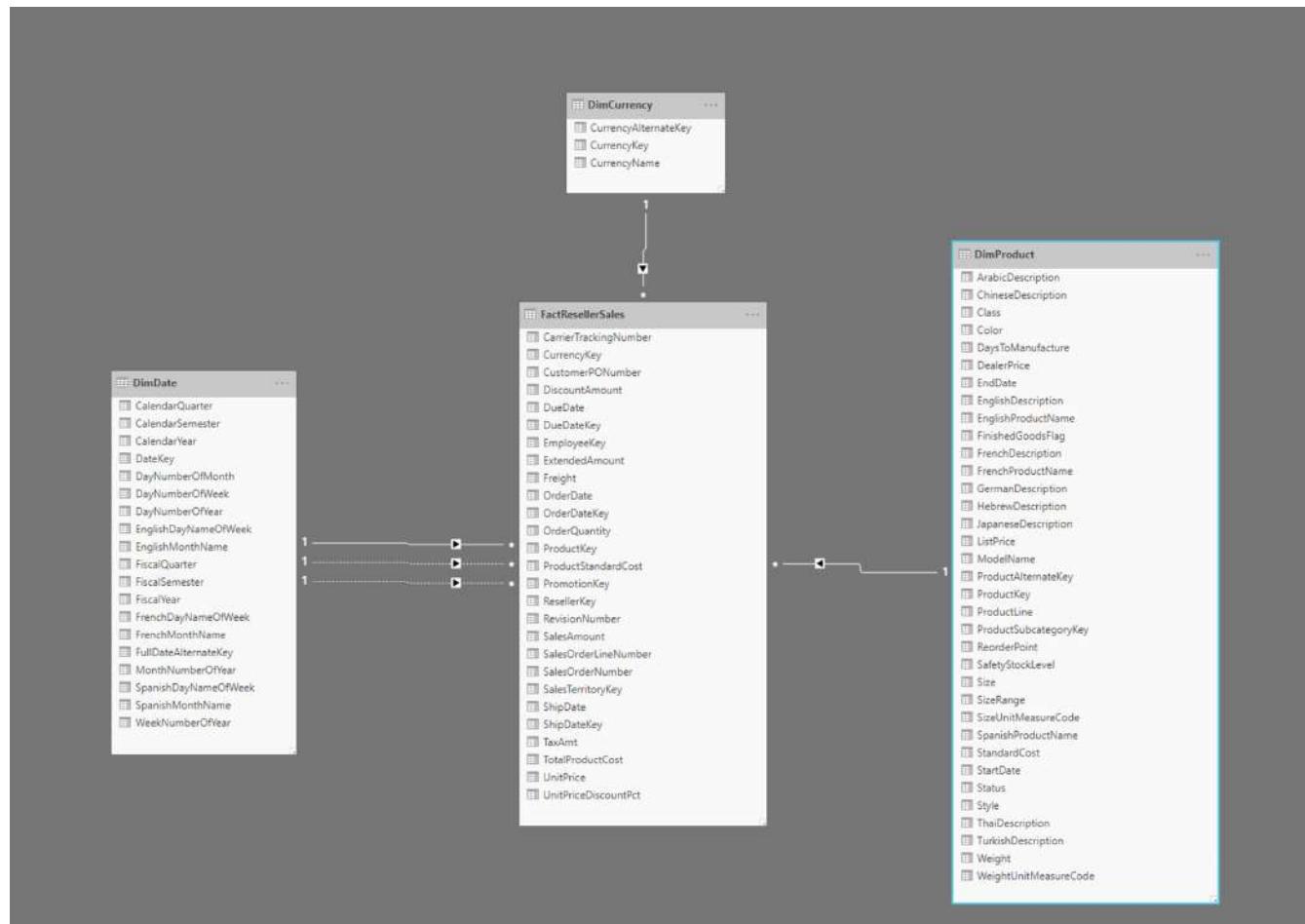
While the third normal form is theoretically desirable, it isn't always possible for all data. In addition, a normalized database doesn't always give you the best performance. Normalized data frequently requires multiple join operations to get all the necessary data returned in a single query. There's a tradeoff between normalizing data when the number of joins required to return query results has high CPU utilization, and denormalized data that has fewer joins and less CPU required, but opens up the possibility of update anomalies.

NOTE: Denormalized data is not the same as unnormalized. For denormalization, we start by designing tables that are normalized. Then we can add additional columns to some tables to reduce the number of joins required, but as we do so, we are aware of the possible update anomalies. We then make sure we have triggers or other kinds of processing that will make sure that when we perform an update, all the duplicate data is also updated.

Denormalized data can be more efficient to query, especially for read heavy workloads like a data warehouse. In those cases, having extra columns may offer better query patterns and/or more simplistic queries.

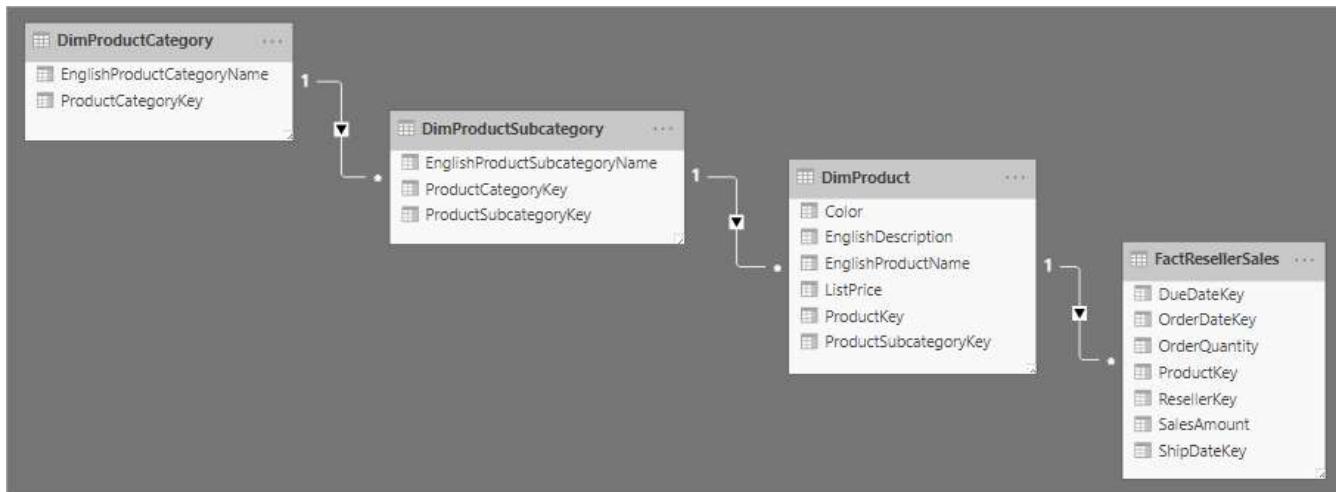
Star schema

While most normalization is aimed at OLTP workloads, data warehouses have their own modeling structure, which is usually a **denormalized** model. This design uses fact tables, which record measurements or metrics for specific events like a sale, and joins them to dimension tables, which are smaller in terms of row count, but may have a large number of columns to describe the fact data. Some example dimensions would include inventory, time, and/or geography. This design pattern is used to make the database easier to query and offer performance gains for read workloads.



The above image shows an example of a star schema, including a *FactResellerSales* fact table, and dimensions for date, currency, and products. The fact table contains data related to the sales transactions, and the dimensions only contain data related to a specific element of the sales data. For example, the *FactResellerSales* table contains only a *ProductKey* to indicate which product was sold. All of the details about each product are stored in the *DimProduct* table, and related back to the fact table with the *ProductKey* column.

Related to star schema design is a snowflake schema, which uses a set of more normalized tables for a single business entity. The following image shows an example of a single dimension for a snowflake schema. The Products dimension is normalized and stored in three tables called *DimProductCategory*, *DimProductSubcategory*, and *DimProduct*.



The main difference between star and snowflake schemas is that the dimensions in a snowflake schema are normalized to reduce redundancy, which saves storage space. The tradeoff is that your queries require more joins, which can increase your complexity and decrease performance.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Choose appropriate data types

SQL Server offers a wide variety of data types to choose from, and your choice can affect performance in many ways. While SQL Server can convert some data types automatically (we call this an ‘implicit conversion’, conversion can be costly and can also negatively affect query plans. The alternative is an explicit conversion, where you use the CAST or CONVERT function in your code to force a data type conversion.

Additionally, choosing data types that are much larger than needed can cause wasted space and require more pages than is necessary to be read. It’s important to choose the right data types for a given set of data—which will reduce the total storage required for the database and improve the performance of queries executed.

NOTE: In some cases, conversions are not possible at all. For example, a date cannot be converted to a bit. Conversions can negatively impact query performance by causing index scans where seeks would have been possible, and additional CPU overhead from the conversion itself.

The image below indicates in which cases SQL Server can do an implicit conversion and in which cases you must explicitly convert data types in your code.

From	To	binary	varbinary	char	varchar	nchar	nvarchar	datetime	smalldatetime	date	time	datetimeoffset	datetime2	decimal	numeric	float	real	bign	int(INT4)	smallint(INT2)	tinyint(INT1)	money	smallmoney	bit	timestamp	uniqueidentifier	ntext	text	sql_variant	xml	CLR UDT	hierarchyid
		●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
binary	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
varbinary	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
char	binary	■	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
varchar	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
nchar	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
nvarchar	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
datetime	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
smalldatetime	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
date	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
time	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
datetimeoffset	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
datetime2	binary	■	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
decimal	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
numeric	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
float	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
real	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
bign	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
int(INT4)	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
smallint(INT2)	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
tinyint(INT1)	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
money	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
smallmoney	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
bit	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
timestamp	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
uniqueidentifier	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
image	binary	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
ntext	binary	✗	✗	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
text	binary	✗	✗	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		
sql_variant	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		
xml	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		
CLR UDT	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		
hierarchyid	binary	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		

■ Explicit conversion
 ● Implicit conversion
 ✗ Conversion not allowed
 ♦ Requires explicit CAST to prevent the loss of precision or scale that might occur in an implicit conversion.
 ○ Implicit conversions between xml data types are supported only if the source or target is untyped xml. Otherwise, the conversion must be explicit.

SQL Server offers a set of system supplied data types for all data that can be used in your tables and queries. SQL Server allows the creation of user defined data types in either T-SQL or the .NET framework.

This document below

Design indexes

SQL Server has several index types to support different types of workloads. At a high level, an index can be thought of as an on-disk structure that is associated with a table or a view, that enables SQL Server to more easily find the row or rows associated with the index key (which consists of one or more columns in the table or view), compared to scanning the entire table.

Clustered indexes

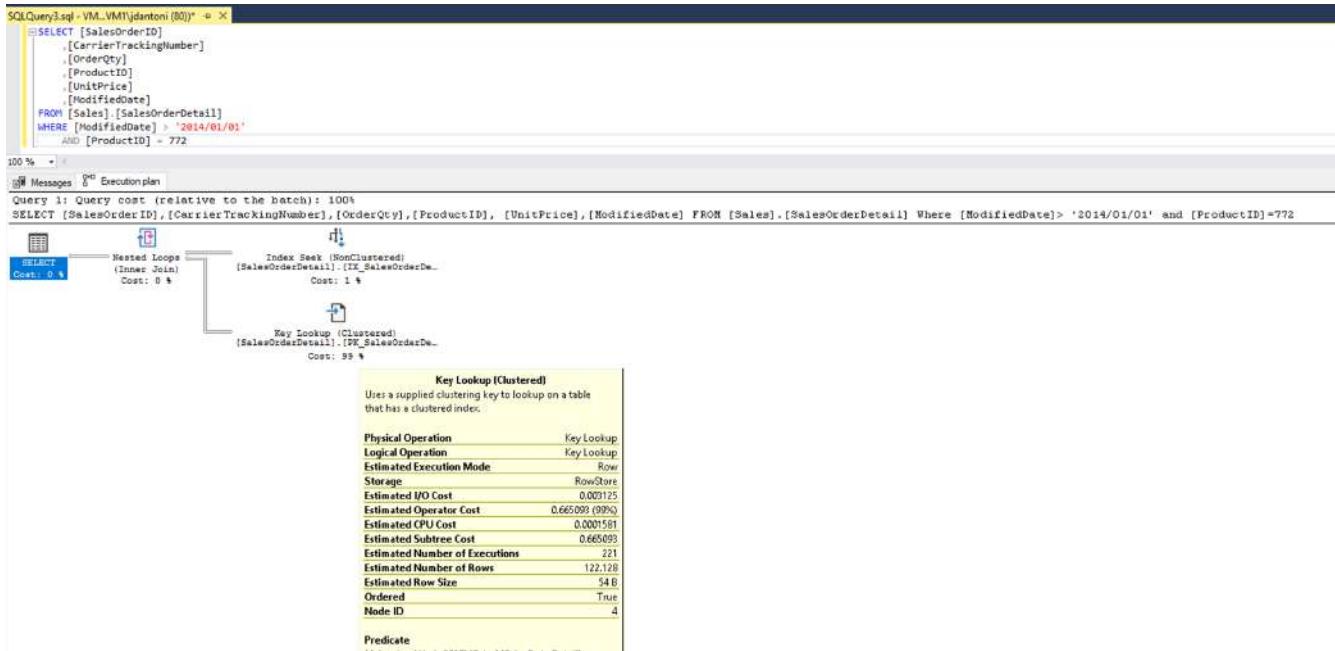
A common DBA job interview question is to ask the candidate the difference between a clustered and nonclustered index, as indexes are a fundamental data storage technology in SQL Server. A clustered index is the underlying table, stored in sorted order based on the key value. There can only be one clustered index on a given table, because the rows can be stored in one order. A table without a clustered index is called a heap, and heaps are typically only used as staging tables. An important performance design principle is to keep your clustered index key as narrow as possible. When considering the key column(s) for your clustered index, you should consider columns that are unique or that contain many distinct values. Another property of a good clustered index key is for records that are accessed sequentially, and are used frequently to sort the data retrieved from the table. Having the clustered index on the column used for sorting can prevent the cost of sorting every time that query executes, because the data will be already stored in the desired order.

NOTE: When we say that the table is 'stored' in a particular order, we are referring to the logical order, not necessarily the physical, on-disk order. Indexes have pointers between pages, and the pointers help create the logical order. When scanning an index 'in order', SQL Server follows the pointers from page to page. Immediately after creating an index, it is most likely also stored in physical order on the disk, but after you start making modifications to the data, and new pages need to be added to the index, the pointers will still give us the correct logical order, but the new pages will most likely not be in physical disk order.

Nonclustered indexes

Nonclustered indexes are a separate structure from the data rows. A nonclustered index contains the key values defined for the index, and a pointer to the data row that contains the key value. You can add other nonkey columns to the leaf level of the nonclustered index to cover more columns using the included columns feature in SQL Server. You can create multiple nonclustered indexes on a table.

An example of when you need to add an index or add columns to an existing nonclustered index is shown below:



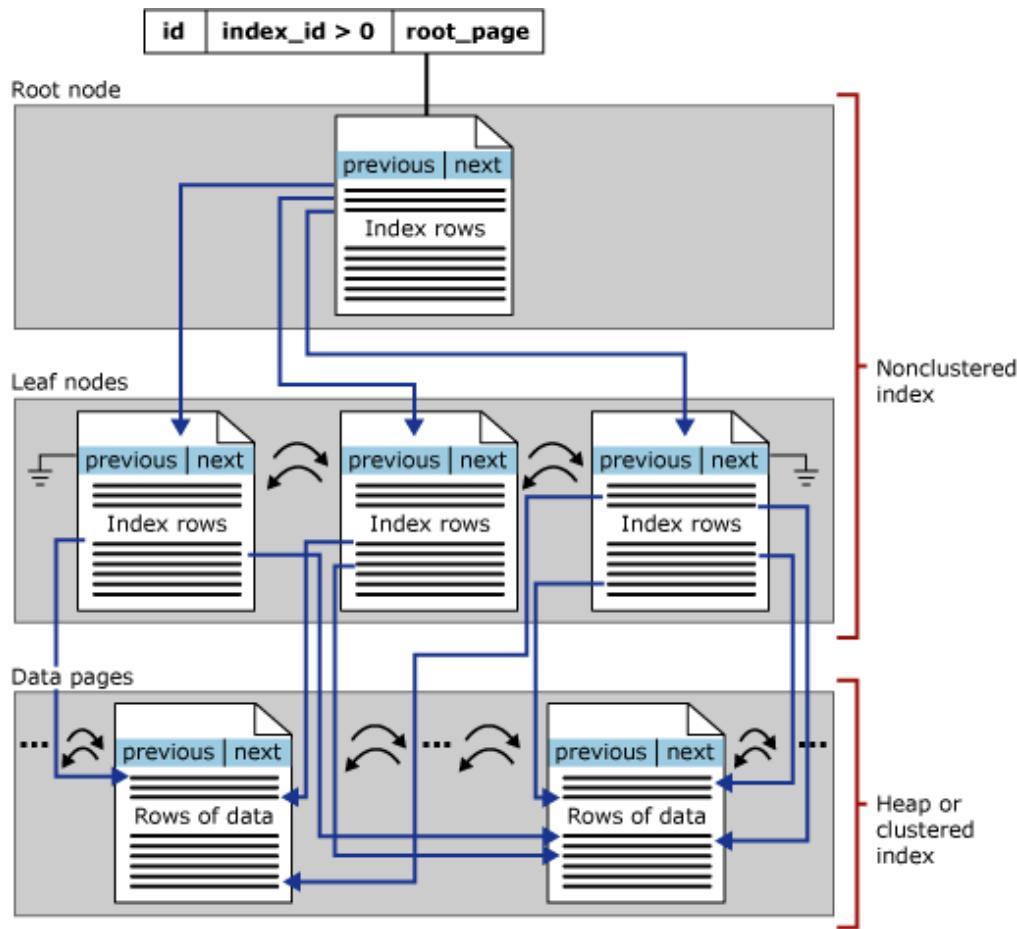
The query plan indicates that for each row retrieved using the index seek, more data will need to be retrieved from the clustered index (the table itself). There's a nonclustered index, but it only includes the product column. If you add the other columns in the query to a nonclustered index as shown below, you can see the execution plan change to eliminate the key lookup.

The index created above is an example of a covering index, where in addition to the key column you're including extra columns to cover the query and eliminate the need to access the table itself.

Both nonclustered and clustered indexes can be defined as unique, meaning there can be no duplication of the key values. Unique indexes are automatically created when you create a PRIMARY KEY or UNIQUE constraint on a table.

The focus of this section is on b-tree indexes in SQL Server—these are also known as row store indexes. The general structure of a b-tree is shown below:





Each page in an index b-tree is called an index node, and the top node of b-tree is called the root node. The bottom nodes in an index are called leaf nodes and the collection of leaf nodes is the leaf level.

Index design is a mixture of art and science. A narrow index with few columns in its key requires less time to update and has lower maintenance overhead; however it may not be useful for as many queries as a wider index that includes more columns. You may need to experiment with several indexing approaches based on the columns selected by your application's queries. The query optimizer will generally choose what it considers to be the best existing index for a query; however, that doesn't mean that there isn't a better index that could be built.

Properly indexing a database is a complex task. When planning your indexes for a table, you should keep a few basic principles in mind:

- Understand the workloads of the system. A table that is used mainly for insert operations will benefit far less from extra indexes than a table used for data warehouse operations that are 90% read activity.
- Understand what queries are run most frequently, and optimize your indexes around those queries
- Understand the data types of the columns used in your queries. Indexes are ideal for integer data types, or unique or non-null columns.
- Create nonclustered indexes on columns that are frequently used in predicates and join clauses, and keep those indexes as narrow as possible to avoid overhead.
- Understand your data size/volume – A table scan on a small table will be a relatively cheap operation and SQL Server may decide to do a table scan simply because it's easy (trivial) to do. A table scan on a large table would be costly.

Another option SQL Server provides is the creation of filtered indexes. Filtered indexes are best suited to columns in large tables where a large percentage of the rows has the same value in that column. A practical example would be an employee table, as shown below, that stored the records of all employees, including ones who had left or retired.

```
CREATE TABLE [HumanResources].[Employee]
(
    [BusinessEntityID] [int] NOT NULL,
    [NationalIDNumber] [nvarchar](15) NOT NULL,
    [LoginID] [nvarchar](256) NOT NULL,
    [OrganizationNode] [hierarchyid] NULL,
    [OrganizationLevel] AS ([OrganizationNode].[GetLevel]()),
    [JobTitle] [nvarchar](50) NOT NULL,
    [BirthDate] [date] NOT NULL,
    [MaritalStatus] [nchar](1) NOT NULL,
    [Gender] [nchar](1) NOT NULL,
    [HireDate] [date] NOT NULL,
    [SalariedFlag] [bit] NOT NULL,
    [VacationHours] [smallint] NOT NULL,
    [SickLeaveHours] [smallint] NOT NULL,
    [CurrentFlag] [bit] NOT NULL,
    [rowguid] [uniqueidentifier] ROWGUIDCOL NOT NULL,
    [ModifiedDate] [datetime] NOT NULL" title="" target="_blank" data-
generated='''>Employee</a>
```

In this table, there's a column called CurrentFlag, which indicates if an employee is currently employed. This example uses the bit datatype, indicating only two values, one for currently employed and zero for not currently employed. A filtered index with a WHERE CurrentFlag = 1, on the CurrentFlag column would allow for efficient queries of current employees.

You can also create indexes on views, which can provide significant performance gains when views contain query elements like aggregations and/or table joins.

Columnstore indexes

Columnstore offers improved performance for queries that run large aggregation workloads. This type of index was originally targeted at data warehouses, but over time columnstore indexes have been used in many other workloads in order to help solve query performance issues on large tables. As of SQL Server 2014, there are both nonclustered and clustered columnstore indexes. Like b-tree indexes, a clustered columnstore index is the table itself stored in a special way, and nonclustered columnstore indexes are stored independently of the table. Clustered columnstore indexes inherently include all the columns in a given table. However, unlike rowstore clustered indexes, clustered columnstore indexes are NOT sorted.

Nonclustered columnstore indexes are typically used in two scenarios, the first is when a column in the table has a data type that isn't supported in a columnstore index. Most data types are supported but XML, CLR, sql_variant, ntext, text, and image aren't supported in a columnstore index. Since a clustered columnstore always contains all the columns of the table (because it IS the table), a nonclustered is the only option. The second scenario is a filtered index—this scenario is used in an architecture called hybrid transactional analytic processing (HTAP), where data is being loaded into the underlying table, and at the same time reports are being run on the table. By filtering the index (typically on a date field), this design allows for both good insert and reporting performance.

Columnstore indexes are unique in their storage mechanism, in that each column in the index is stored independently. It offers a two-fold benefit. A query using a columnstore index only needs to scan the columns

needed to satisfy the query, reducing the total IO performed, and it allows for greater compression, since data in the same column is likely to be similar in nature.

Columnstore indexes perform best on analytic queries that scan large amounts of data, like fact tables in a data warehouse. Starting with SQL Server 2016 you can augment a columnstore index with another b-tree nonclustered index, which can be helpful if some of your queries do lookups against singleton values.

Columnstore indexes also benefit from batch execution mode, which refers to processing a set of rows (typically around 900) at a time versus the database engine processing those rows one at time. Instead of loading each record independently and processing them, the query engine computes the calculation in that group of 900 records. This processing model reduces the number of CPU instructions dramatically.

```
SELECT SUM(Sales) FROM SalesAmount;
```

Batch mode can provide significant performance increase over traditional row processing. SQL Server 2019 also includes batch mode for rowstore data. While batch mode for rowstore doesn't have the same level of read performance as a columnstore index, analytical queries may see up to a 5x performance improvement.

The other benefit columnstore indexes offer to data warehouse workloads is an optimized load path for bulk insert operations of 102,400 rows or more. While 102,400 is the minimum value to load directly into the columnstore, each collection of rows, called a rowgroup, can be up to approximately 1,024,000 rows. Having fewer, but fuller, rowgroups makes your SELECT queries more efficient, because fewer row groups need to be scanned to retrieve the requested records. These loads take place in memory and are directly loaded to the index. For smaller volumes, data is written to a b-tree structure called a delta store, and asynchronously loaded into the index.

In this example, the same data is being loaded into two tables, *FactResellerSales_CCI_Demo* and *FactResellerSales_Page_Demo*. The *FactResellerSales_CCI_Demo* has a clustered columnstore index, and the *FactResellerSales_Page_Demo* has a clustered b-tree index with two columns and is page compressed. As you can see each table is loading 1,024,000 rows from the *FactResellerSalesXL_CCI* table. When `SET STATISTICS TIME` is ON, SQL Server keeps track of the elapsed time of the query execution. Loading the data into the columnstore table took roughly 8 seconds, where loading into the page compressed table took nearly 20 seconds. In this example, all the rows going into the columnstore index are loaded into a single rowgroup.

If you load less than 102,400 rows of data into a columnstore index in a single operation, it's loaded in b-tree structure known as a delta store. The database engine moves this data into the columnstore index using an asynchronous process called the tuple mover. Having open delta stores can affect the performance of your queries, because reading those records is less efficient than reading from the columnstore. You can also reorganize the index with the `COMPRESS_ALL_ROW_GROUPS` option in order to force the delta stores to be added and compressed into the columnstore indexes.

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What type of database design should you use for a data warehouse when you want to reduce the data volume of your dimensions?

- Snowflake schema
- Star Schema
- Third normal form

Check Answers

Multiple choice

What is the minimum number of rows you need to bulk insert into a columnstore index?

- 102,400
- 1,000,000
- 1000

Check Answers

Multiple choice

Which index type is recommended for queries that run large aggregation workloads?

- Columnstore index
- Clustered index
- Nonclustered index

Check Answers

Summary

Database design is a critical component of overall performance. The SQL Server platform offers a number of options for building indexes and compressing data for better performance. Using the proper data types and storage mechanisms can also help the performance of your applications.

Now that you've reviewed this module, you should be able to:

- Understand normal form and how it affects database design
- Choose appropriate datatypes for your database
- Understand types of indexes

Introduction

One of the challenges of the DBA's role is to evaluate the changes they make to code or data structures on a busy, production system. While tuning a single query in isolation offers easy metrics such as elapsed time or logical reads, making minor tweaks on a busy system may require deeper evaluation.

Learning objectives:

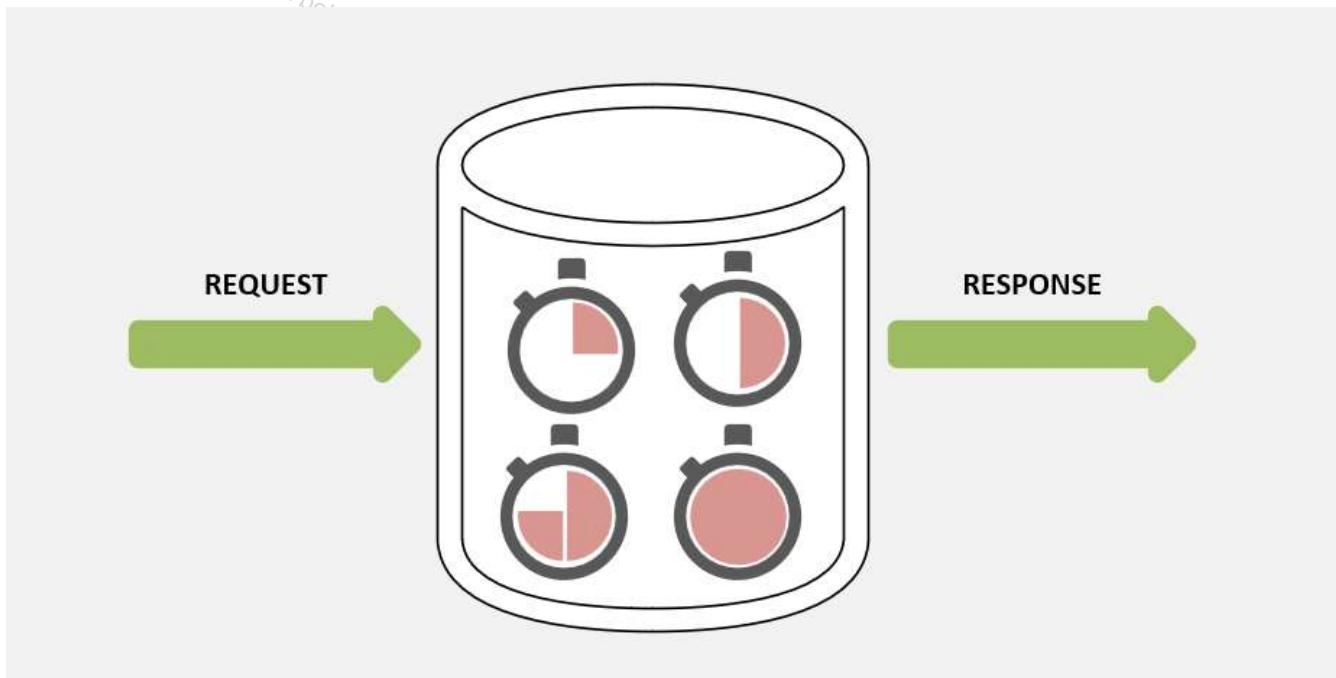
At the end of this module, you'll be able to:

- Describe wait statistics
- Understand important aspects when tuning indexes
- Explore when to use query hints

Describe wait statistics

One holistic way of monitoring server performance is to evaluate what the server is waiting on. Wait statistics are complex, and SQL Server is instrumented with hundreds of waiting types, which monitors each running thread and logs what the thread is waiting on.

Detecting and troubleshooting SQL Server performance issues require an understanding of how wait statistics work, and how the database engine uses them while processing a request.



Wait statistics are broken down into three types of waits: **resource waits**, **queue waits**, and **external waits**.

- **Resource waits** occur when a worker thread in SQL Server requests access to a resource that is currently being used by a thread. Examples of resources wait are locks, latches, and disk I/O waits.
- **Queue waits** occur when a worker thread is idle and waiting for work to be assigned. Example queue waits are deadlock monitoring and deleted record cleanup.
- **External waits** occur when SQL Server is waiting on an external process like a linked server query to complete. An example of an external wait is a network wait related to returning a large result set to a client application.

You can check `sys.dm_os_wait_stats` system view to explore all the waits encountered by threads that executed, and `sys.dm_db_wait_stats` for Azure SQL Database. The `sys.dm_exec_session_wait_stats` system view lists active waiting sessions.

These system views allow the DBA to get an overview of the performance of the server, and to readily identify configuration or hardware issues. This data is persisted from the time of instance startup, but the data can be cleared as needed to identify changes.

Wait statistics are evaluated as a percentage of the total waits on the server.

	WaitType	Wait Percentage	AvgWait_Sec
1	REDO_THREAD_PENDING_WORK	24.75	0.1016
2	CXPACKET	17.19	0.0021
3	CXCONSUMER	12.24	0.0090
4	PARALLEL_REDO_TRAN_TURN	10.60	0.0029
5	SOS_SCHEDULER_YIELD	10.06	0.0022
6	BPSORT	3.80	0.0126
7	PAGEIOLATCH_SH	3.60	0.0029
8	BACKUPIO	2.01	0.0110
9	HTDELETE	1.83	0.1573
10	LATCH_EX	1.81	0.0047

The result of this query from `sys.dm_os_wait_stats` shows the wait type, and the aggregation of percent of time waiting (*Wait Percentage* column) and the average wait time in seconds for each wait type.

In this case, the server has Always On Availability Groups in place, as indicated by the **REDO_THREAD_PENDING_WORK** and **PARALLEL_REDO_TRAN_TURN** wait types. The relatively high percentage of **CXPACKET** and **SOS_SCHEDULER_YIELD** waits indicates that this server is under some CPU pressure.

As DMVs provide a list of wait types with the highest time accumulated since the last SQL Server startup, collecting and storing wait statistic data periodically could help you understand and correlate performance problems with other database events.

Considering that DMVs provide you with a list of wait types with the highest time accumulated since the last SQL Server startup, collecting and storing wait statistics periodically might help you understand and correlate performance problems with other database events.

There are several types of waits available in SQL Server, but some of them are common.

- **RESOURCE_SEMAPHORE**—this wait type is indicative of queries waiting on memory to become available, and may indicate excessive memory grants to some queries. This problem is typically observed by long query runtimes or even time outs. These wait types can be caused by out-of-date statistics, missing indexes, and excessive query concurrency.
- **LCK_M_X**—frequent occurrences of this wait type can indicate a blocking problem, that can be solved by either changing to the `READ COMMITTED SNAPSHOT` isolation level, or making changes in indexing to reduce transaction times, or possibly better transaction management within T-SQL code.
- **PAGEIOLATCH_SH**—this wait type can indicate a problem with indexes (or a lack of useful indexes), where SQL Server is scanning too much data. Alternatively, if the wait count is low, but the wait time is high, it can indicate storage performance problems. You can observe this behavior by analyzing the data in the `waiting_tasks_count` and the `wait_time_ms` columns in the `sys.dm_os_wait_stats` system view, to calculate an average wait time for a given wait type.
- **SOS_SCHEDULER_YIELD**—this wait type can indicate high CPU utilization, which is correlated with either high number of large scans, or missing indexes, and often with high numbers of **CXPACKET** waits.

- **CXPACKET**—if this wait type is high it can indicate improper configuration. Prior to SQL Server 2019, the max degree of parallelism default setting is to use all available CPUs for queries. Additionally, the cost threshold for parallelism setting defaults to 5, which can lead to small queries being executed in parallel, which can limit throughput. Lowering MAXDOP and increasing the cost threshold for parallelism can reduce this wait type, but the **CXPACKET** wait type can also indicate high CPU utilization, which is typically resolved through index tuning.
- **PAGEIOLATCH_UP**—this wait type on data pages 2:1:1 can indicate TempDB contention on Page Free Space (PFS) data pages. Each data file has one PFS page per 64 MB of data. This wait is typically caused by only having one TempDB file, as prior to SQL Server 2016 the default behavior was to use one data file for TempDB. The best practice is to use one file per CPU core up to eight files. It's also important to ensure your TempDB data files are the same size and have the same autogrowth settings to ensure they're used evenly. SQL Server 2016 and higher control the growth of TempDB data files to ensure they grow in a consistent, simultaneous fashion.

In addition to the aforementioned DMVs, the Query Store also tracks waits associated with a given query. However, waits data tracked by Query Store isn't tracked at the same granularity as the data in the DMVs, but it can provide a nice overview of what a query is waiting on.

Tune and maintain indexes

The most common (and most effective) method for tuning T-SQL queries is to evaluate and adjust your indexing strategy. Properly indexed databases perform fewer IOs to return query results, and with fewer IOs there's reduced pressure on both the IO and storage systems. Reducing IO even allows for better memory utilization. Keep in mind the read/write ratio of your queries.

A heavy write workload may indicate that the cost of writing rows to extra indexes isn't of much benefit. An exception would be if the workload performs mainly updates that also need to do *lookup* operations. Update operations that do lookups can benefit from extra indexes or columns added to an existing index. Your goal should always be to get the most benefit out of the smallest number of indexes on your tables.

A common performance tuning approach is as follows:

- Evaluate existing index usage using `sys.dm_db_index_operational_stats` and `sys.dm_db_index_usage_stats`.
- Consider eliminating unused and duplicate indexes, but this should be done carefully. Some indexes may only be used during monthly/quarterly/annual operations, and may be important for those processes. You may also consider creating indexes to support those operations just before the operations are scheduled, to reduce the overhead of having otherwise unused indexes on a table.
- Review and evaluate expensive queries from the Query Store, or Extended Events capture, and work to manually craft indexes to better serve those queries.
- Create the index(s) in a non-production environment, and test query execution and performance and observe performance changes. It's important to note any hardware differences between your production and non-production environments, as the amount of memory and the number of CPUs could affect your execution plan.
- After testing carefully, implement the changes to your production system.

Verify the column order of your indexes—the leading column drives column statistics and usually determines whether the optimizer will choose the index. Ideally, the leading column will be selective and used in the `WHERE` clause of many of your queries. Consider using a change control process for tracking changes that could affect application performance. Before dropping an index, save the code in your source control, so the index can be quickly recreated if an infrequently run query requires the index to perform well.

Finally, columns used for **equality comparisons** should precede columns used for **inequality comparisons** and that columns with greater selectivity should precede columns with fewer distinct values.

Resumable index

Resumable index allows index maintenance operations to be paused, or take place in a time window, and be resumed later. A good example of where to use resumable index operations is to reduce the impact of index maintenance in a busy production environment. You can then perform rebuild operations during a specific maintenance window giving you more control over the process.

Furthermore, creating an index for a large table can negatively affect the performance of the entire database system. The only way to fix this issue in versions prior to SQL Server 2019 is to kill the index creation process. Then you have to start the process over from the beginning if the system rolls back the session.

With resumable index, you can pause the build and then restart it later at the point it was paused.

The following example shows how to create a resumable index:

```
-- Creates a nonclustered index for the Customer table

CREATE INDEX IX_Customer_PersonID_ModifiedDate
    ON Sales.Customer (PersonID, StoreID, TerritoryID, AccountNumber, ModifiedDate)
WITH (RESUMABLE=ON, ONLINE=ON)
GO
```

In a query window, resume the index operation:

```
ALTER INDEX IX_Customer_PersonID_ModifiedDate ON Sales.Customer PAUSE
GO
```

The statement above uses the **PAUSE** clause to temporarily stop the creation of the resumable online index.

You can check the current execution status for a resumable online index by querying the **sys.index_resumable_operations** system view.

NOTE: Resumable index is only supported with online operations.

Understand query hints

Query hints are options or strategies that can be applied to enforce the query processor to use a particular operator in the execution plan for **SELECT**, **INSERT**, **UPDATE**, or **DELETE** statements. Query hints override any execution plan the query processor might select for a given query with the **OPTION** clause.

In most cases, the query optimizer selects an efficient execution plan based on the indexes, statistics, and data distribution. Database administrators rarely need to intervene manually.

You can change the execution plan of the query by adding query hints to the end of the query. For example, if you add **OPTION (MAXDOP <integer_value>)** to the end of a query that uses a single CPU, the query may use multiple CPUs (parallelism) depending on the value you choose. Or, you can use **OPTION (RECOMPILE)** to ensure that the query generates a new, temporary plan each time it's executed.

```
--with maxdop hint
SELECT ProductID, OrderQty, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
GROUP BY ProductID, OrderQty
ORDER BY ProductID, OrderQty
OPTION (MAXDOP 2)
GO

--with recompile hint
SELECT City
FROM Person.Address
WHERE StateProvinceID=15 OPTION (RECOMPILE)
GO
```

Although query hints may provide a localized solution to various performance-related issues, you should avoid using them in production environment for the following reasons.

- Having a permanent query hint on your query can result in structural database changes that would be beneficial to that query not being applicable.
- You can't benefit from new and improved features in subsequent versions of SQL Server if you bind a query to a specific execution plan.

However, there are several query hints available on SQL Server, which are used for different purposes. Let's discuss a few of them below:

- **FAST <integer_value>** —retrieves the first <integer_value> number of rows while continuing query execution. It works better with small data sets and low value for fast query hint. As row count is increased, query cost becomes higher.
- **OPTIMIZE FOR** —provides instructions to the query optimizer that a particular value for a local variable should be used when a query is compiled and optimized.
- **USE PLAN** —the query optimizer will use a query plan specified by the *xml_plan* attribute.
- **RECOMPILE** —creates a new, temporary plan for the query and discards it immediately after the query is executed.

- `{ LOOP | MERGE | HASH } JOIN`—specifies all join operations are performed by `LOOP JOIN`, `MERGE JOIN`, or `HASH JOIN` in the whole query. The optimizer chooses the least expensive join strategy from among the options if you specify more than one join hint.
- `MAXDOP <integer_value>`—overrides the max degree of parallelism value of `sp_configure`. The query specifying this option also overrides the Resource Governor.

You can also apply multiple query hints in the same query. The following example uses the `HASH GROUP` and `FAST <integer_value>` query hints in the same query.

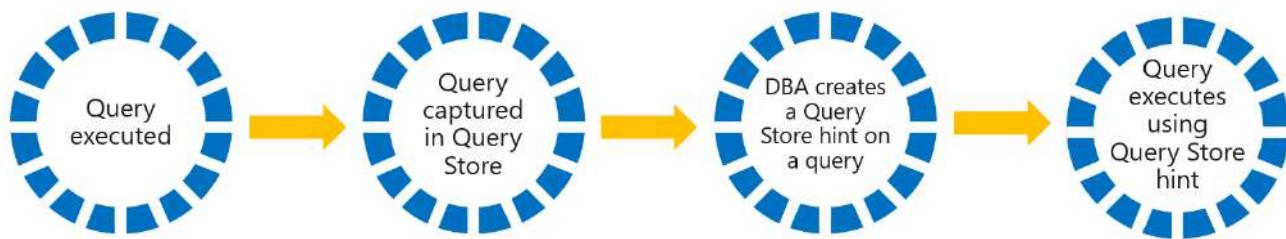
```
SELECT ProductID, OrderQty, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
WHERE UnitPrice < $5.00
GROUP BY ProductID, OrderQty
ORDER BY ProductID, OrderQty
OPTION (HASH GROUP, FAST 10);
GO
```

To learn more about query hints, see [Hints \(Transact-SQL\)](#).

Query Store hints (in preview)

The Query Store hints feature in Azure SQL Database provides a simple method for shaping query plans without modifying application code.

Query Store hints are useful when the query optimizer doesn't generate an efficient execution plan, and when the developer or DBA can't modify the original query text. In some applications, the query text may be hardcoded or automatically generated.



To use Query Store hints, you need to identify the Query Store `query_id` of the query statement you wish to modify through Query Store catalog views, built-in Query Store reports, or Query Performance Insight for Azure SQL Database. Then, execute `sp_query_store_set_hints` with the `query_id` and query hint string you wish to apply to the query.

The example below shows how to obtain the `query_id` for a specific query, and then use it to apply the `RECOMPILE` and `MAXDOP` hints to the query.

```
SELECT q.query_id, qt.query_sql_text
FROM sys.query_store_query_text qt
INNER JOIN sys.query_store_query q
ON qt.query_text_id = q.query_text_id
WHERE query_sql_text like N'%ORDER BY CustomerName DESC%'
AND query_sql_text not like N'%query_store%'
GO
```

```
--Assuming the query_id returned by the previous query is 42  
EXEC sys.sp_query_store_set_hints @query_id= 42, @query_hints = N'OPTION(RECOMPILE, MAXDOP  
1)'  
GO
```

There are a few scenarios where Query Store hints can help with query-level performance issues.

- Recompile a query on each execution.
- Limit the maximum degree of parallelism for a statistic update operation.
- Use a Hash join instead of a Nested Loops join.
- Use compatibility level 110 for a specific query while keeping the database at the current compatibility.

NOTE: Query Store hints are also supported by SQL Managed Instance.

For more information about Query Store hints, see [Query Store hints](#).

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What type of index is best used on a data warehouse fact table?

- Clustered Columnstore
- Nonclustered Columnstore
- Clustered b-tree

Check Answers

Multiple choice

Which DMV provides information about server level wait statistics?

- sys.dm_db_index_physical_stats
- sys.dm_os_wait_stats
- sys.dm_exec_session_wait_stats

Check Answers

Multiple choice

Which DMV can you use to capture the last Actual Execution Plan for a given query?

- sys.dm_exec_cached_plans
- sys.dm_exec_query_plan
- sys.dm_exec_query_plan_stats

Check Answers

Summary

One of the challenges of the DBA's role is to evaluate the impact of changes they make to code or data structures on a busy, production system. While tuning a single query in isolation offers easy metrics such as elapsed time or logical reads, making minor tweaks on a busy system may require deeper evaluation.

Azure SQL is a deeply instrumented software and allows for monitoring at several levels.

Learning objectives:

Now that you've reviewed this module, you should be able to:

- Describe wait statistics
- Understand important aspects when tuning indexes
- Explore when to use query hints

Introduction

On-premises systems require cabling and racks of hardware in order to deploy a new database server. With cloud computing, this isn't necessary. One of the major benefits of cloud computing is that system resources are abstracted behind an API.

A common term used around cloud computing deployments is *infrastructure as code*, which means all of your resources are defined a set of scripts that are stored in source control and can easily be deployed to a new environment. Although stateful resources don't need to be deployed as frequently as application code, by defining your infrastructure, you ensure that resources are implemented consistently, reducing configuration risk and human error.

Learning objectives

After completing this module, you will be able to:

- Describe the deployment models available on Azure
- Deploy database resources using PowerShell and Azure CLI
- Deploy an Azure Resource Manager template and Bicep
- Understand the difference between multiple command-line options

Describe deployment models in Azure

Azure Resource Manager templates have the benefit of being able to deploy a full set of resources in one single declarative template. You can build dependencies into the templates, and using parameters to change deployment values at deployment time. Once you have a template, you can deploy it several ways including using an Azure DevOps pipeline, or through the custom deployments options in the Azure portal. The benefit of these deployments is that they use a declarative model, which defines what should be created. The Azure Resource Manager framework then determines how to deploy it. The alternative to the declarative model is the imperative model. Imperative frameworks include PowerShell and the Azure CLI, which follow a prescriptive order of tasks to be executed.

As mentioned above, there are two programming models that are used for cloud deployments: **imperative** and **declarative**.

In **imperative** programming, you're defining a set of prescriptive tasks for the target system to execute. A simple example of this model is using a batch script to install SQL Server and its prerequisites. In **declarative** programming, you're describing a set of resources to be defined, typically by a framework. A simple example is a CREATE TABLE statement, which describes the columns and keys to be built by the SQL Server engine. The statement acts as the framework for building the table.

Azure Resource Manager templates

Azure Resource Manager templates allow you to create and deploy an entire infrastructure in a declarative framework. For example, you can deploy not only a virtual machine, but its network and storage dependencies in one document. Resource Manager also supports orchestration, which manages the deployment of interdependent resources so that they're created in the correct order. For example, a VM is dependent on the existence of a virtual network, so the framework will deploy the network (or check for the existence of the network) before attempting to build the VM. Azure Resource Manager templates also support extensibility, which allows you to run PowerShell or Bash scripts on your resources after they're deployed.

PowerShell

PowerShell provides a core module known as Az PowerShell module, which has child resource providers for nearly all Azure services. For example, **Az.Compute** would cover Azure Virtual Machines. PowerShell is more commonly used for resource modification and status retrieval. While it's possible to create resources using PowerShell, it isn't typically used for complex deployments. PowerShell can also be used to deploy Azure Resource Manager templates, so in a sense it supports both declarative and imperative models.

Azure CLI

The Azure Command Line Interface, or CLI, is similar to PowerShell in that it can be used either imperatively or declaratively. Much like PowerShell and Azure Resource Manager templates, the Azure CLI provides a mechanism to deploy or modify Azure Resources. Some commands for Azure PostgreSQL and Azure MySQL Databases are only available in the Azure CLI.

Azure portal

The Azure portal is a graphical interface to Azure Resource Manager. Any resources you build and deploy using the portal will have an Azure Resource Manager template that you can capture by selecting **Export template** in the **Overview** section of your resource group.

The screenshot shows the Azure portal's resource group overview for 'contoso_rg'. The 'Essentials' section displays basic information: Subscription (Contoso Subscription), Subscription ID (54545q22-3356-4e27-b768-1172cd01srw), Tags (Click here to add tags), Deployments (3 Failed, 42 Succeeded), and Location (West US). Below this, the 'Resources' section lists a single item: 'AdventureWorksLT (dp500-east-sql/AdventureWorksLT)' which is a 'SQL database' located in 'East US'. The portal interface includes a left sidebar with navigation links like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Deployments, Security, Policies, Properties, and Locks. The top navigation bar includes options like Create, Manage view, Delete resource group, Refresh, Export to CSV, Open query, Assign tags, Move, Delete, and Export template.

The Azure portal is usually the easiest way to get started when first learning about the Azure platform. Organizations (and DBAs) typically move into a more automated model of deployment as their Azure estate and experience grows.

Azure DevOps

In Azure DevOps, deployments are carried out using Azure Pipelines. Azure Pipelines are a fully featured continuous integration and continuous delivery service (CI/CD), which allows you to automate the build, testing, and deployment of your code. You can deploy Azure resources using Azure Resource Manager templates in two ways... The first method calls a PowerShell script as shown above. The second approach defines tasks that stage your artifacts (the templates themselves and any required secrets) and then deploys the templates. One task stages the artifacts and the other tasks deploys the templates.

Continuous integration

Continuous integration is a development methodology that focused on making small changes to code and frequent code check-ins to the version control system. Continuous integration provides an automated way to build, package, and test applications. Having the framework in place facilitates more frequent check-ins, and allows better collaboration between developers, with the goal of improving code quality. Continuous delivery builds on the continuous integration and automates the delivery of code changes to the underlying infrastructure.

Automate deployment by using Azure Resource Manager templates and Bicep

Automating database deployment is a crucial ability to create a reliable and sustainable development process. This module will also provide you with the information to be able to use ARM templates and bicep files in your database deployments.

ARM template

Azure Resource Manager (ARM) templates are JavaScript Object Notation (JSON) documents that describe the resources to deploy within an Azure Resource Group. ARM templates are declarative, and allow you to specify your resources and properties without having to write a full sequence of programming commands.

ARM templates allow you to create and deploy your entire infrastructure using a declarative framework. For example, you can deploy not only a virtual machine, but its network and storage dependencies in a single document. ARM templates also support orchestration, which manages the deployment of interdependent resources so that they're created in the correct order and extensibility, which allows you to run PowerShell or Bash scripts after you've deployed your resources.

Benefits

ARM templates offer the following benefits:

- **Repeatable** – ARM templates are *idempotent*, which means it allows you to repeatedly deploy your infrastructure throughout the development lifecycle and have confidence your resources are deployed in a consistent manner.
 - **Orchestration** – ARM templates take care of the complexities of ordering operations for deployments and when possible will deploy resources in parallel rather than serial for faster deployments.
 - **Modular** – ARM templates can be split and combined at will, so you can create the deployments you need.
 - **Exportable code** – A great way to learn the template syntax is to export the current template. Exporting templates allow you to easily recreate your environment for Disaster Recovery or documentation purposes.
 - **Authoring tools** – ARM templates can be authored using the freely available Visual Studio Code and the template tool extension. It provides intellisense, syntax highlighting, in-line help, and many other language functions. In addition to Visual Studio Code, you can also use Visual Studio.

Deploy an ARM template with PowerShell

You have several options for the scope of your deployment when using PowerShell and ARM templates. You can deploy to a resource group, a subscription, a Management Group (a collection of subscriptions under the same Azure template and commonly used in large enterprise deployments), or a tenant.

Let's look at a JSON ARM template definition to create a single database in SQL Database:

```
{  
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",
```

```
"metadata": {  
    "_generator": {  
        "name": "bicep",  
        "version": "0.5.6.12127",  
        "templateHash": "17606057535442789180"  
    }  
},  
"parameters": {  
    "serverName": {  
        "type": "string",  
        "defaultValue": "[uniqueString('sql', resourceGroup().id)]",  
        "metadata": {  
            "description": "The name of the SQL logical server."  
        }  
    },  
    "sqlDBName": {  
        "type": "string",  
        "defaultValue": "SampleDB",  
        "metadata": {  
            "description": "The name of the SQL Database."  
        }  
    },  
    "location": {  
        "type": "string",  
        "defaultValue": "[resourceGroup().location]",  
        "metadata": {  
            "description": "Location for all resources."  
        }  
    },  
    "administratorLogin": {  
        "type": "string",  
        "metadata": {  
            "description": "The administrator username of the SQL logical server."  
        }  
    },  
    "administratorLoginPassword": {  
        "type": "secureString",  
        "metadata": {  
            "description": "The administrator password of the SQL logical server."  
        }  
    }  
},  
"resources": [  
    {  
        "type": "Microsoft.Sql/servers",  
        "apiVersion": "2021-08-01-preview",  
        "name": "[parameters('serverName')]",  
        "location": "[parameters('location')]",  
        "properties": {  
            "administratorLogin": "[parameters('administratorLogin')]",  
            "administratorLoginPassword": "[parameters('administratorLoginPassword')]"  
        }  
    },  
    {  
        "type": "Microsoft.Sql/servers/databases",  
        "dependsOn": "[resourceId('Microsoft.Sql/servers', parameters('serverName'))]"  
    }  
]
```

```

    "apiVersion": "2021-08-01-preview",
    "name": "[format('{0}/{1}', parameters('serverName'), parameters('sqlDBName'))]",
    "location": "[parameters('location')]",
    "sku": {
        "name": "Standard",
        "tier": "Standard"
    },
    "dependsOn": [
        "[resourceId('Microsoft.Sql/servers', parameters('serverName'))]"
    ]
}
]
}
}

```

As we can see above, a single database that has one of two purchasing models was defined. As part of creating a single database, you also specify the server to be responsible for managing it, and the region in which it should be placed within Azure.

As we can see in the PowerShell example below, this file can be deployed from a URL:

```

$ projectName = Read-Host -Prompt "Enter a project name that is used for generating resource names"
$ location = Read-Host -Prompt "Enter an Azure location (i.e. centralus)"
$ adminUser = Read-Host -Prompt "Enter the SQL server administrator username"
$ adminPassword = Read-Host -Prompt "Enter the SQL server administrator password" -AsSecureString

$ resourceGroupName = "${projectName}rg"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -Templateuri
"https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.sql/sql-database/azuredeploy.json" -
administratorLogin $adminUser -administratorLoginPassword $adminPassword

```

Bicep

Azure Bicep is a declarative language that allows you to deploy Azure resources. Bicep provides a first-class authoring experience that is concise, reliable and allow for code reuse, and it's commonly described as an Infrastructure-as-Code (IaC) tool.

Bicep isn't meant to be a general programming language. It's meant as a tool to allow you to create a file declaring Azure infrastructure resources and properties that can be used throughout the development lifecycle, allowing resource deployment in a consistent manner.

Benefits

The following are some benefits of Bicep:

- Continuous full support** – Bicep provides support for all resource types and API versions for Azure services, which means that as soon as a resource provider introduces new resource types and API versions, you can use them in your Bicep file without waiting for a tool update.

- **Simple syntax** – Compared to an equivalent JSON file, Bicep files will be more concise and easier to read.
- **Easy to use:** Bicep requires no previous knowledge of programming languages and is easy to write and understand.

The following examples show the difference between a Bicep file and the equivalent JSON template. Both examples deploy a storage account.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]"
    },
    "storageAccountName": {
      "type": "string",
      "defaultValue": "[format('toylaunch{0}', uniqueString(resourceGroup().id))]"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2021-06-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {
        "accessTier": "Hot"
      }
    }
  ]
}
```

```
param location string = resourceGroup().location
param storageAccountName string = 'toylaunch${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
```

```

    }
}

```

Bicep vs. JSON

Both Bicep and JSON can be used to deploy a database. As the example above shows, Bicep is a lot more concise and simpler to read. Here's an example of a JSON file to deploy a database:

```

{
  "type": "Microsoft.Sql/servers",
  "apiVersion": "2021-11-01-preview",
  "name": "string",
  "location": "string",
  "tags": {
    "tagName1": "tagvalue1",
    "tagName2": "tagvalue2"
  },
  "identity": {
    "type": "string",
    "userAssignedIdentities": {}
  },
  "properties": {
    "administratorLogin": "string",
    "administratorLoginPassword": "string",
    "administrators": {
      "administratorType": "ActiveDirectory",
      "azureADOnlyAuthentication": "bool",
      "login": "string",
      "principalType": "string",
      "sid": "string",
      "tenantId": "string"
    },
    "federatedClientId": "string",
    "keyId": "string",
    "minimalTlsVersion": "string",
    "primaryUserAssignedIdentityId": "string",
    "publicNetworkAccess": "string",
    "restrictOutboundNetworkAccess": "string",
    "version": "string"
  }
}

```

Now, compare it to a Bicep file:

```

resource symbolicname 'Microsoft.Sql/servers@2021-11-01-preview' = {
  name: 'string'
  location: 'string'
  tags: {
    tagName1: 'tagvalue1'
    tagName2: 'tagvalue2'
  }
  identity: {
    type: 'string'
  }
}

```

```

    userAssignedIdentities: {}
}
properties: {
    administratorLogin: 'string'
    administratorLoginPassword: 'string'
    administrators: {
        administratorType: 'ActiveDirectory'
        azureADOnlyAuthentication: bool
        login: 'string'
        principalType: 'string'
        sid: 'string'
        tenantId: 'string'
    }
    federatedClientId: 'string'
    keyId: 'string'
    minimalTlsVersion: 'string'
    primaryUserAssignedIdentityId: 'string'
    publicNetworkAccess: 'string'
    restrictOutboundNetworkAccess: 'string'
    version: 'string'
}
}
}

```

As you can see, the Bicep file is easier to read and use. You can also install the Bicep extension for Visual Studio Code to create your Bicep files, as the editor provides rich intellisense, and syntax validation.

Deploying an Azure SQL Database using Bicep with PowerShell

You can easily create an Azure SQL Database using Bicep and PowerShell. A single database has a defined set of compute, memory, IO, and storage resources using one of two purchasing models. When you create a single database, you also define a server to manage it and place it within Azure resource group in a specified region. Below is the respective Bicep file.

The Bicep file used in this quickstart is from [Azure Quickstart Templates](#).

```

@description('The name of the SQL logical server.')
param serverName string = uniqueString('sql', resourceGroup().id)

@description('The name of the SQL Database.')
param sqlDBName string = 'SampleDB'

@description('Location for all resources.')
param location string = resourceGroup().location

@description('The administrator username of the SQL logical server.')
param administratorLogin string

@description('The administrator password of the SQL logical server.')
@secure()
param administratorLoginPassword string

```

```
resource sqlServer 'Microsoft.Sql/servers@2021-08-01-preview' = {
    name: serverName
    location: location
    properties: {
        administratorLogin: administratorLogin
        administratorLoginPassword: administratorLoginPassword
    }
}

resource sqlDB 'Microsoft.Sql/servers/databases@2021-08-01-preview' = {
    parent: sqlServer
    name: sqlDBName
    location: location
    sku: {
        name: 'Standard'
        tier: 'Standard'
    }
}
```

To deploy this file, save it as main.bicep on your local computer and run the following commands in PowerShell.

```
New-AzResourceGroup -Name exampleRG -Location eastus
New-AzResourceGroupDeployment -ResourceGroupName exampleRG -TemplateFile ./main.bicep
```

Source control for templates

ARM templates and Bicep files are an example of infrastructure as code. Since all hardware resources are abstracted behind a set of APIs, your entire infrastructure can just be another component of your application code. Just like application or database code, it's important to protect and version this code. In addition to the internal version in the template, your source control system should version your templates.

In most cases, the database administrator won't be writing their own template from scratch. You may either build them from the Azure portal or using a template from the [quickstart templates](#) that are provided by Microsoft on GitHub.

The image below shows how to create a SQL Database from a template on GitHub.

This screenshot shows a GitHub repository page for a Bicep template. The repository path is `azure-quickstart-templates / quickstarts / microsoft.sql / sql-database /`. The 'Deploy to Azure' button is highlighted with a red box.

Files

- README.md
- azuredeploy.json
- azuredatabase.parameters.json
- main.bicep
- metadata.json

Commits

File	Description	Date
README.md	Update API and bicep (#12126)	5 months ago
azuredeploy.json	update to bicep v0.5.6 (#12409)	2 months ago
azuredatabase.parameters.json	MOVE: 101-sql-database to quickstarts/microsoft.sql/sql-database	14 months ago
main.bicep	provider updates and bicep functionality (#12232)	4 months ago
metadata.json	provider updates and bicep functionality (#12232)	4 months ago

Actions

- Azure Public Test Date: 2022.05.12 | Azure Public Test Result: pass
- Azure US Gov Test Date: 2022.05.12 | Azure US Gov Test Result: pass
- Best Practice Check: pass | CredScan Check: Not Tested | Bicep Version: 0.6.11
- Deploy to Azure** (highlighted with a red box)
- Deploy to Azure Gov
- Visualize

This template allows you to create an [Azure SQL Database](#). To learn more about how to deploy the template, see the [quickstart article](#).

Tags: Azure, SQL database

Select **Deploy to Azure**, and you'll need to log into the Azure portal.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Create a SQL Server and Database

...

Azure quickstart template

[Basics](#) [Review + create](#)

Template



sql-database ↗

2 resources

[Edit template](#)[Edit parameters](#)[Visualize](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Contoso Subscription

Resource group * ⓘ

contoso_rg

[Create new](#)

Instance details

Region * ⓘ

(US) West US

Server Name ⓘ

dp300-sqlserver

Sql DB Name ⓘ

SampleDB

Location ⓘ

Administrator Login * ⓘ

sqladmin

Administrator Login Password * ⓘ

.....

[Review + create](#)[< Previous](#)[Next : Review + create >](#)

If you select **Edit template**, you'll see the JSON defining the template, which would allow you to change the values to meet your requirements.

After providing the required parameters in the deployment screen, select **Review + create** to deploy your template.

Automate deployment by using PowerShell

PowerShell is a modern, cross-platform command shell that simplifies task management and provides powerful features for automation. It was created using widely used languages to provide administrators with command-line features, which, when used with automation, supports the reduction of ongoing operational cost.

PowerShell can accept and return both text and .NET objects that allow it to be a dynamic, one-stop command-line tool.

Some of the many benefits of PowerShell include:

- Robust command-line history
- Tab completion and command prediction
- Supports command and parameter aliases
- Pipeline for chaining commands
- In-console help system

PowerShell provides a core module known as Az PowerShell module, which is an open-source set of cmdlets that replaces AzureRM and are used to manage Azure resources directly from PowerShell. The Az PowerShell module allows for Azure resource creation, modification, status retrieval, and template-based deployments.

Az.Sql PowerShell module

The Az.Sql PowerShell module is a subset of Az PowerShell that allows you to manage and deploy Azure SQL resources. Everything from the creation of a database to configuring geo replication to full Azure SQL management can be accomplished with the Az.Sql cmdlets.

The Az.Sql PowerShell module can be used however you use PowerShell including PowerShellGet, the Azure Cloud Shell and an Az PowerShell Docker container.

Regardless of how you use PowerShell, the syntax used is still the same with the verb-noun structure as seen below:

```
<command-name> -<Required Parameter Name> <Required Parameter value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<optional Parameter Name>] <Required Parameter value>
```

Commands always begin with a command name, such as `Get-AzSqlServer`, which returns information about one or more Azure SQL Database servers. The “command-name” is then followed by a Parameter Name with <-ServerName> being an applicable parameter for Get-AzSqlServer. This is then followed with a parameter value, which is written in a string form. Below is an example usage of the `Get-AzSqlServer` command with multiple parameters with its return values:

```
Get-AzSqlServer -ResourceGroupName "ResourceGroup01" -ServerName "Server01"
```

You can see below a few other examples like how to create a new SQL Managed Instance, and how to create a database on a specific server:

```
New-AzSqlInstance -Name managedInstance2 -ResourceGroupName ResourceGroup01 -Location westcentralus -AdministratorCredential (Get-Credential) -SubnetId "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx/resourceGroups/resourcegroup01/providers/Microsoft.Network/virtualNetworks/vnet" -LicenseType LicenseIncluded -StorageSizeInGB 1024 -VCore 16 -Edition "GeneralPurpose" -ComputeGeneration Gen4
```

```
New-AzSqlDatabase -ResourceGroupName "ResourceGroup01" -ServerName "Server01" -DatabaseName "Database01"
```

Here's an example that creates a new SQL Managed Instance with External Azure Active Directory administrator, Azure Active Directory Only authentication and no `SqlAdministratorCredentials`:

```
New-AzSqlInstance -Name managedInstance2 -ResourceGroupName ResourceGroup01 -ExternalAdminName DummyLogin -EnableActiveDirectoryOnlyAuthentication -Location westcentralus -SubnetId "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx/resourceGroups/resourcegroup01/providers/Microsoft.Network/virtualNetworks/vnet" -LicenseType LicenseIncluded -StorageSizeInGB 1024 -VCore 16 -Edition "GeneralPurpose" -ComputeGeneration Gen4
```



```
$val = Get-AzSqlInstance -Name managedInstance2 -ResourceGroupName ResourceGroup01 -ExpandActiveDirectoryAdministrator
```

To learn more about the full list of command-names for the Az.Sql module, see [Azure PowerShell Az.Sql](#).

Automate deployment by using Azure CLI

Database automation is now no longer reserved for larger businesses, but is becoming a necessity for businesses of all sizes to remain competitive. As a database administrator, it's important to learn how to automate database tasks whenever possible as it will provide you with the following benefits:

- Granular control of an application or database
- Easy to scale, improving efficiency when dealing with large numbers of assets
- Ability to reuse scripts to automate regular tasks
- Facilitates troubleshooting tasks where GUI tools aren't available

The Azure Command-Line Interface (CLI) is a cross-platform command-line tool that helps you to create and manage Azure resources. You can run commands through the terminal using interactive command-line prompts or scripts.

You can install Azure CLI on Linux, Mac, or Windows computers. Run it from a browser using the Cloud Shell terminal on Azure portal or inside a Docker container.

The Azure CLI syntax follows the `reference name - command - parameter - parameter value` pattern. For example, switching between subscriptions is often a common task. Here's the syntax.

```
az account set --subscription "my subscription name"
```

PowerShell vs. Azure CLI

Azure PowerShell and Azure CLI are both cross-platform command-line tools that will enable you to create and manage Azure resources on Windows, macOS and Linux. The main difference between the two is the shell environments that they support.

Shell Environment	Azure CLI	Azure Powershell
Cmd	Yes	
Bash	Yes	
Windows PowerShell	Yes	Yes
PowerShell	Yes	Yes

To choose the correct tool, consider your experience and work environment.

The Azure CLI is similar to bash scripting, and it will feel natural to those who typically work with Linux systems. Azure PowerShell contains modules that help manage Azure resources from PowerShell. PowerShell commands follow the standard verb-noun syntax, and working with Windows systems will make it a natural fit.

Here's a quick comparison of some commonly used commands in both their CLI and PowerShell forms:

Command	Azure CLI	Azure PowerShell
Sign in with Web Browser	az login	Connect-AzAccount
Get available subscriptions	az account list	Get-AzSubscription
Set Subscription	az account set --subscription	Set-AzContext -Subscription
List VM	az vm list	Get-AzVM
Create a SQL Server	az sql server create	New-AzSqlServer

Deploying SQL Database using Azure CLI

Below is an example of how to deploy SQL Database, and create a firewall rule that allows access from Azure services using Azure CLI.

```
let "randomIdentifier=$RANDOM*$RANDOM"

$resourceGroup = "<your resource group>"
.setLocation = "<your location preference>"
$server = "dp300-sql-server-$randomIdentifier"
$login = "sqladmin"
$password = "Pa$$w0rD-$randomIdentifier"

az sql server create --name $server --resource-group $resourceGroup --location "$location"
az sql server firewall-rule create --resource-group $resourceGroup --server $server
```

To learn more about all the Azure SQL CLI commands available, see [Azure SQL CLI commands](#).

Deploying Azure Resource Manager (ARM) template using Azure CLI and PowerShell

With PowerShell, you have several options for the scope of your deployment. You can deploy to a resource group, a subscription, a Management Group (a collection of subscriptions under the same Azure template and commonly used in large enterprise deployments), or a tenant. Azure Resource Manager templates are parameterized, and you will need to pass in parameters, either inline or through the use of a parameter file as shown in the example below.

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName
ExampleResourceGroup
-TemplateFile c:\MyTemplates\azuredeploy.json
-TemplateParameterFile c:\MyTemplates\storage.parameters.json
```

The parameter and template file can also be stored in a Git repo, Azure Blob Storage, or any other place where it is accessible from the deploying machine.

Azure CLI allows the same options for deployment scope as you have with PowerShell. Like with PowerShell, you can use a local or remote parameter file and template, as shown in the example below.

```
az deployment group create --resource-group ExampleResourceGroup --template-file '\p
```

To deploy remote linked templates with relative path that are stored in a storage account, use query-string to specify the SAS token:

```
az deployment group create \  
  --name linkedTemplateWithRelativePath \  
  --resource-group myResourceGroup \  
  --template-uri "https://stage20210126.blob.core.windows.net/template-staging/main" \  
  --query-string $sasToken
```

NOTE: Currently, Azure CLI doesn't support deploying remote Bicep files. You can use Bicep CLI to build the Bicep file to a JSON template, and then load the JSON file to the remote location.

You can review your deployed resources in Azure CLI using the command below:

```
az resource list --resource-group exampleRG
```

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What language are Azure Resource Manager templates written in?

- JSON
- C#
- T-SQL

Check Answers

Multiple choice

Which option should you include in your template when specifying the region for a resource group deployment?

- Parameter
- Variable
- Output

Check Answers

Multiple choice

Which element of a template allows for you to build dependencies into resources?

- dependsOn
- concat
- apiVersion

Check Answers

Multiple choice

Which Azure CLI command should you choose to create a SQL Database?

- az sql db op
- az sql server create
- az sql db create

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

The Azure platform provides many options for deploying resources. You have the option to deploy either from Azure Pipelines or through command line scripting options. Azure Resource Manager templates and Bicep files provide you the flexibility to granularly deploy Azure resources in a repeatable consistent fashion.

It's considered a best practice to manage your *infrastructure as code* and implement source control for it. This has the side benefit of providing more consistent deployments.

Now that you've reviewed this module, you should be able to:

- To describe the deployment models available on Azure
- How to deploy database resources using PowerShell and CLI
- How to deploy an Azure Resource Manager template and Bicep
- The difference between multiple command-line options

This document belongs to Srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to Srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to Srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Introduction

Database systems need regular maintenance, which includes tasks like making backups and updating statistics. Maintenance may also include regularly scheduled jobs that execute against a database. Some common examples of these jobs would be to extract, transform, and load data from a transaction processing system into a data warehouse. In SQL Server and Azure SQL managed instance, the SQL Server Agent service allows you to schedule jobs to perform these maintenance tasks.

One of the ways you can benefit from Azure is using the built-in resource monitoring that the platform provides. You can also take advantage of the options that the Azure platform offers for handling and responding to events.

Learning objectives

At the end of this module, you will understand:

- What maintenance activities you should perform on your databases
- How to configure notifications and alerts on SQL Server Agent jobs and SQL Server
- How to configure notifications alerts based on performance monitor values

Create a SQL Server maintenance plan

Typical activities that you can schedule for regular SQL Server maintenance include:

- Database and transaction log backups
- Database Consistency Checks
- Index maintenance
- Statistics updates

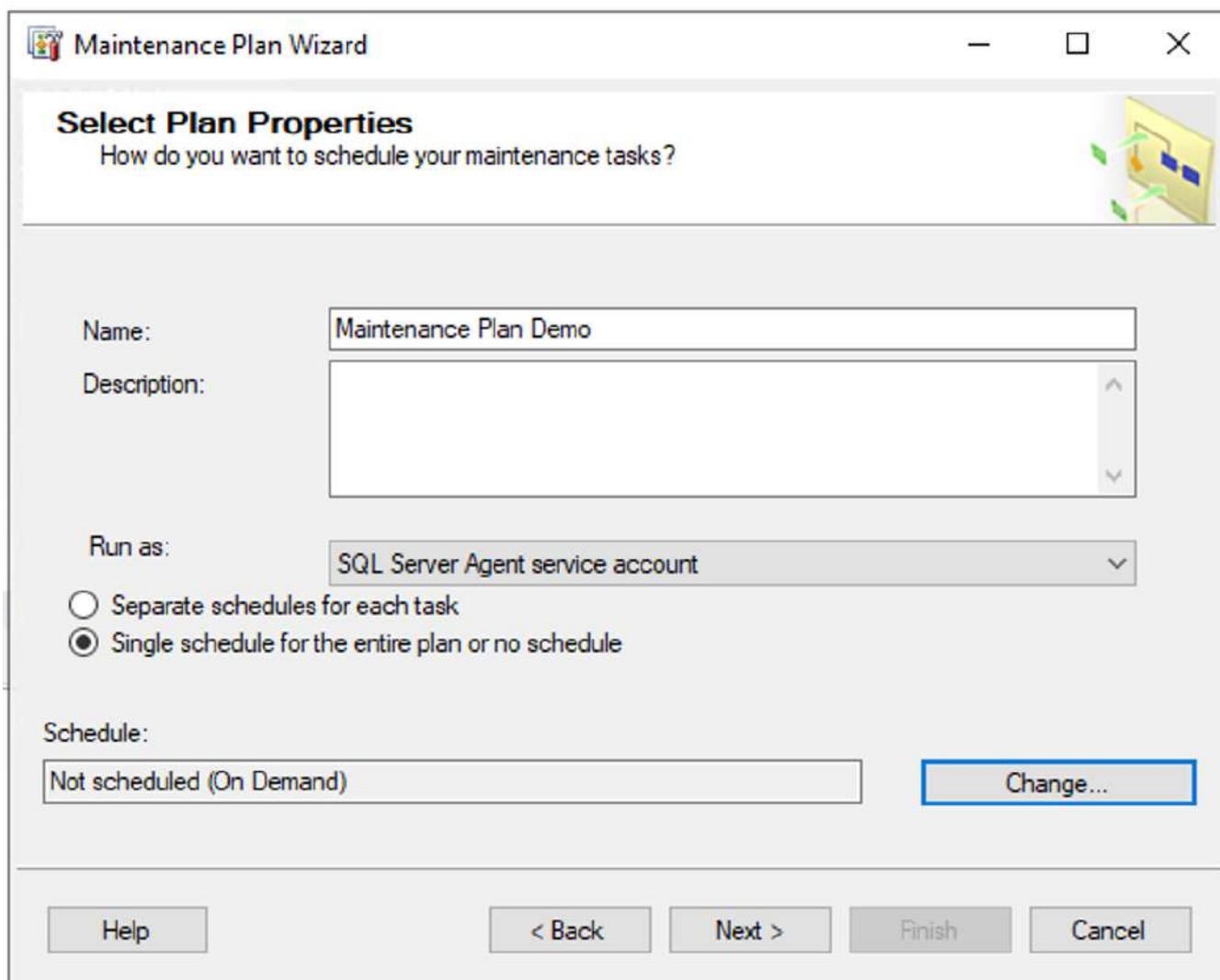
You should be aware of the importance of backups, as well as index and statistics maintenance, for all your databases. Database consistency checks, also known as CHECKDB (for the command DBCC CHECKDB) are of equal importance, as it is the only way to check an entire database for corruption. Depending on the size of your databases and your uptime requirements, you may perform all of these activities nightly. More commonly in production systems, the maintenance operations are spread out over the course of week, as both index maintenance and consistency checks are very I/O intensive operations and they are typically done during weekend hours. Similarly, many DBAs stagger backups of large databases, and only do one full backup a week. Differential and transaction log backups can then be used to manage recovery to a specific point in time. SQL Server offers a built-in way to manage all of these tasks using Maintenance Plans. Maintenance plans create a workflow of the tasks to support your databases. Maintenance plans are created as Integration Services packages, which allow you to schedule your maintenance activities. Many DBAs also use open-source scripts to perform database maintenance, to allow for more flexibility and control of maintenance activities.

Best practices for maintenance plans

In addition to allowing you to perform database maintenance, maintenance plans provide options to allow you to prune data from the `msdb` database, which acts as the data store for the SQL Server Agent. Maintenance plans also allow you to specify that older database backups should be removed from disk. Removing old backup files helps your SQL Server by reducing the size of your backup volume and helps manage the size of the `msdb` database. Ensure that your backup retention period is longer than your consistency check window. This means if you run a consistency check weekly, you should retain eight days of backups. (Note: The backup operation will not detect corruption in a database, so it is possible to have corruption within a backup file). Maintenance plan activities are scheduled as SQL Server Agent jobs for execution.

Creating a maintenance plan

You can create a maintenance plan using SQL Server Management Studio as shown below. Note that in the example below, multiple maintenance tasks are combined in one maintenance plan. The best practice would be to create a maintenance plan for each type of task—and possibly even for a specific database on your server. For example, you might create a maintenance plan to back up system databases and another maintenance plan to back up user databases. You could also have another maintenance plan for special handling of the backup of one very large user database. The image below and the following examples show the creation a maintenance plan using the maintenance plan wizard.



The image above shows the first screen of the maintenance plan wizard from SQL Server Management Studio (SSMS). There are a couple of things you should note — you need to specify a name for your maintenance plan, and you need to specify a run-as account. Typically, most maintenance operations will run as the SQL Server Agent service account, but you may need to have a task run as a different account for security purposes. For example, if you need to back up to a file share that only a specific account has access to, you would need to run that specific plan as a different user. This concept is known as a proxy user and is another component of the SQL Server Agent.

What is a proxy account?

A proxy account is an account with stored credentials that can be used by the SQL Server Agent to execute steps of a job as a specific user. The login information for this user is stored as a credential in the SQL Server instance. Proxy accounts are typically used when very granular security rights are needed for specific steps of a SQL agent job.

Job schedules

Job schedules are a component of the job system in the `msdb` system database. SQL Server Agent jobs and schedules have a many-to-many relationship. Each job can contain multiple schedules, and the same schedule can be assigned to multiple jobs. The maintenance plan wizard, however, does not allow creation of independent schedules. It creates a specific schedule for each maintenance plan, as shown below:

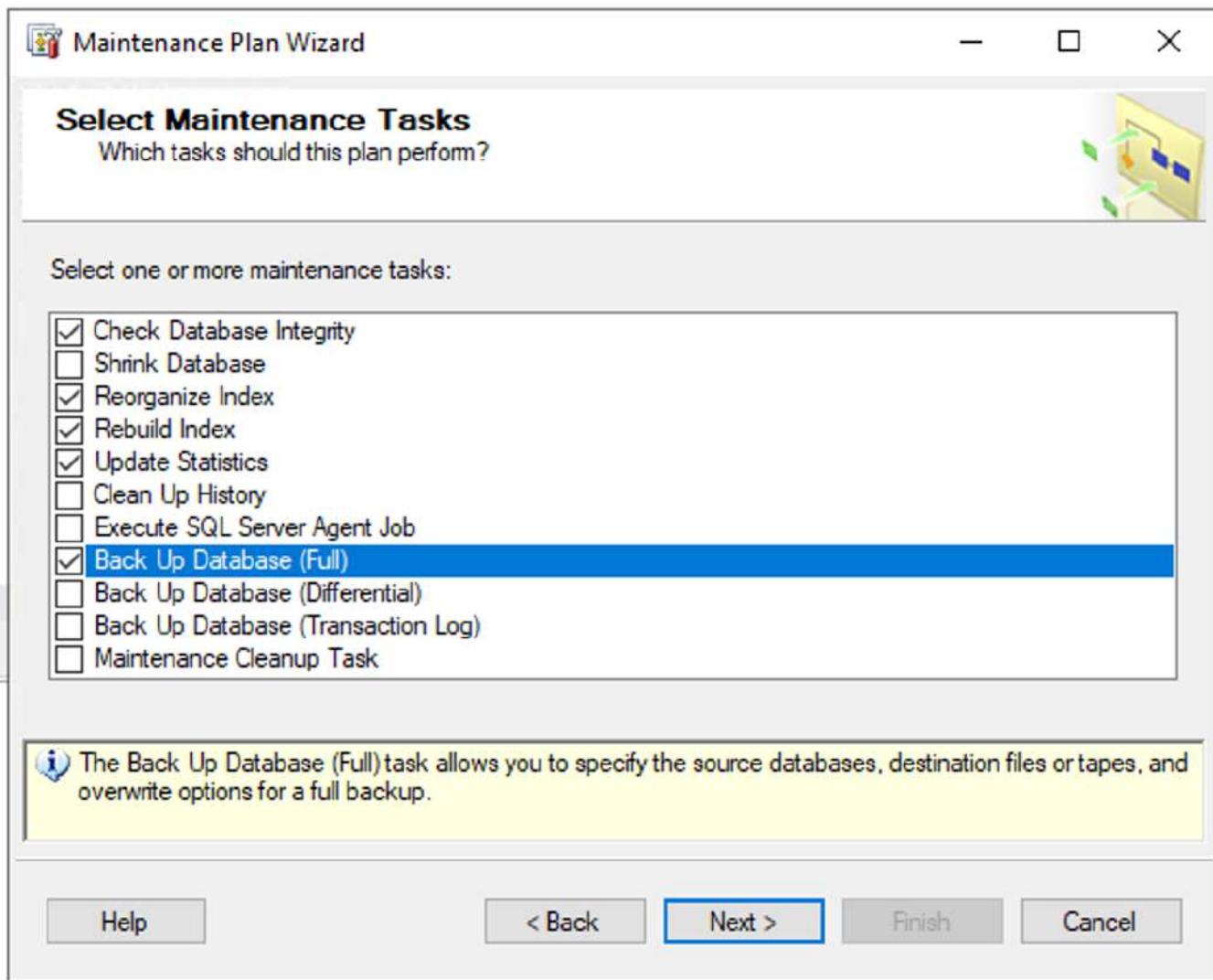
New Job Schedule

Name:	Maintenance Plan Demo	Jobs in Schedule
Schedule type:	Recurring	<input checked="" type="checkbox"/> Enabled
One-time occurrence		
Date:	5/ 5/2020	Time: 9:06:53 AM
Frequency		
Occurs:	Weekly	
Recurs every:	1 week(s) on	
<input checked="" type="checkbox"/> Monday <input type="checkbox"/> Wednesday <input type="checkbox"/> Friday <input type="checkbox"/> Saturday <input type="checkbox"/> Tuesday <input type="checkbox"/> Thursday <input checked="" type="checkbox"/> Sunday		
Daily frequency		
<input checked="" type="radio"/> Occurs once at:	12:00:00 AM	
<input type="radio"/> Occurs every:	1 hour(s)	Starting at: 12:00:00 AM
		Ending at: 11:59:59 PM
Duration		
Start date:	5/ 5/2020	<input type="radio"/> End date: 5/ 5/2020
		<input checked="" type="radio"/> No end date:
Summary		
Description:	Occurs every week on Sunday at 12:00:00 AM. Schedule will be used starting on 5/5/2020.	
<input type="button" value="OK"/> <input type="button" value="Cancel"/> <input type="button" value="Help"/>		

The above schedule is for a weekly execution, but you also have the option to create a schedule with hourly or daily recurrence. The next step in this process is to add maintenance tasks to the plan.

This document belongs to srinivas Ramchand Jillella.
 srinivas9.dba@gmail.com
 No unauthorized copies allowed!

This document bel...



The image above shows the maintenance tasks addition screen. This is where you choose the operations to be performed by your maintenance plan. The options are:

Check Database integrity - This task executes the DBCC CHECKDB command, which validates the contents of each database page to ensure its logical and physical consistency. This task should be performed on a regular basis (daily or weekly), and it should align to your backup retention window. To ensure corruption is not carried over to your backups, make sure you are successfully completing a consistency check before discarding any prior backups.

Shrink Database - This task reduces the size of database or transaction log file by moving data into free space on pages. When enough space is consumed, the free space can be returned to the file system.

NOTE: It is recommended that you never execute this action as part of any regular maintenance as it leads to severe index fragmentation which can harm database performance. The operation itself is also very I/O and CPU intensive and can severely impact your system performance.

Reorganize/Rebuild Index - This task will check the level of fragmentation in a database's indexes, and may either rebuild or reorganize the index based on the user-defined level of fragmentation. Note that rebuilding an index updates the statistics on the index.

Update Statistics - This task updates the column and index statistics that are used by SQL Server to build query execution plans. It is important that the statistics accurately reflect the data stored in tables so that the query optimizer can make the best decisions in building execution plans. This task allows you to choose which tables and

indexes are scanned, and the percentage or number of rows scanned. The default sampling rate is acceptable for most objects, though you may wish to capture more detailed statistics for specific tables.

Cleanup History - This task deletes history of backup and restore operations from the `msdb` database, as well as the history of SQL Server agent jobs. This task is used to manage the size of the `msdb` database.

Execute SQL Server Agent Job - This task is used to execute a user-defined SQL Server Agent job.

Backup Database (Full/Differential/Log) - This task is used to back up databases on a SQL Server instance. A full backup backs up the entire database, and serves as the starting point for a restore (you need a full backup in order to completely restore a database). Differential backups backup the pages in the database that have changed since the last full backup, and are typically used to provide an incremental restore point. Transaction log backups backup the active pages in your transaction log, and allow you to define your recovery point objective. Transaction log backups cannot be performed on databases in SIMPLE recovery mode.

Here's an example of the use of different kinds of backups: If you took a full backup on Sunday, and a differential each week night, and you wanted to restore to your database to noon on Thursday, you would only need to restore Sunday's full backup and Wednesday's differential, followed by the transaction log backups from the point of Wednesday's differential backup until Thursday at noon.

Maintenance Cleanup Tasks - This task removes old files related to maintenance plans, including text reports from maintenance plan execution, and backup files. It only removes backups on files in the folders specified, so any subfolders must be specifically listed or they will be skipped.

Each task has a scope of user databases, system databases, or a custom selection of databases. Additionally, each task has its own specific configuration options.

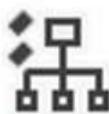
Once you finish creating the Maintenance Plan, you will be presented with the details of the entire plan. You can get back to this view in SQL Server Management Studio by expanding the Management node, then expanding the Maintenance Plans node, right-clicking on this Maintenance Plan and selecting Modify.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!



Check Database Integrity

Check Database integrity on Local server connection
Databases: All databases
Include indexes
Physical only



Reorganize index on Local server connection
Databases: All databases
Object: Tables and views
Compact large objects



Rebuild index on Local server connection
Databases: All databases
Object: Tables and views
Original amount of free space



Update Statistics on Local server connection
Databases: All databases
Object: Tables and views
All existing statistics
Scan type: Full scan



Back Up Database (Full)



Backup Database on Local server connection

Databases: All databases

Type: Full

Append existing

Destination: Disk

Backup Compression (Default)

Upon creation, the plan will appear as a job in the SQL Server Agent. If you added a schedule either during the creation process or after, that job will be executed and the maintenance tasks will be performed.

Multi-server automation

In a multi-server environment, the SQL Server Agent provides the option of designating one server as a master server that can execute jobs on other servers, designated as target servers. The master server stores a master copy of the jobs and distributes the jobs to the target servers. Target servers connect to the master server periodically to update their schedule of jobs. This allows you to define one job and deploy it across your enterprise. A good example of this would be configuring database maintenance across your environment. You could create a set of maintenance plan tasks once and allow them to be pushed out to group of target servers, to ensure consistent deployment.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.

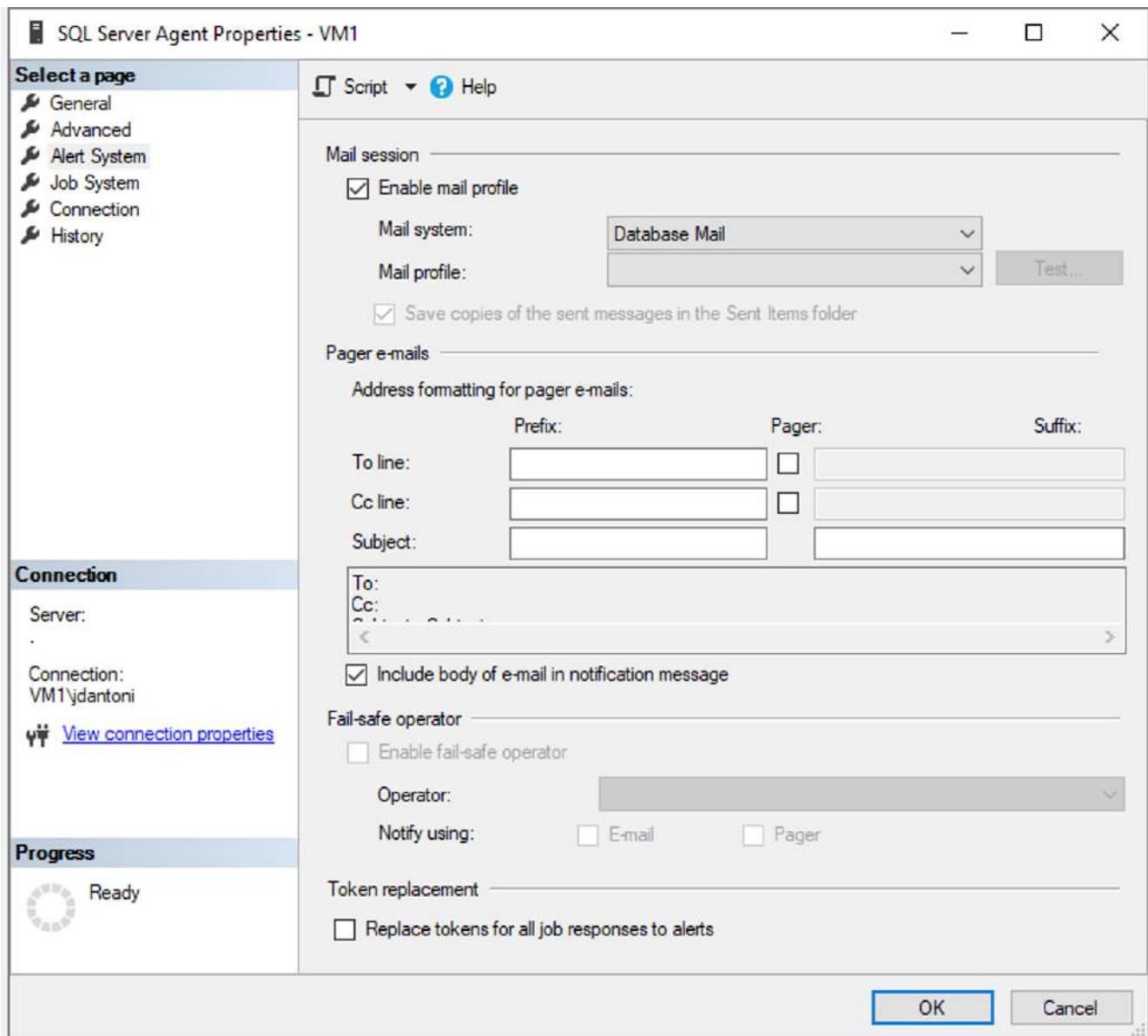
Describe task status notifications

One important part of automation is providing notifications in the event of job failure or if certain system errors are encountered. SQL Server Agent provides this functionality through a group of objects. Alerting is most commonly done via email using the Database Mail functionality of SQL Server. The other agent objects that are used in this workflow are:

- Operators—alias for people or group who receives notifications.
- Notifications—notify an operator of the completion, success, or failure of a job.
- Alerts—are assigned to an operator, for either a notification or a defined error condition.

Operators

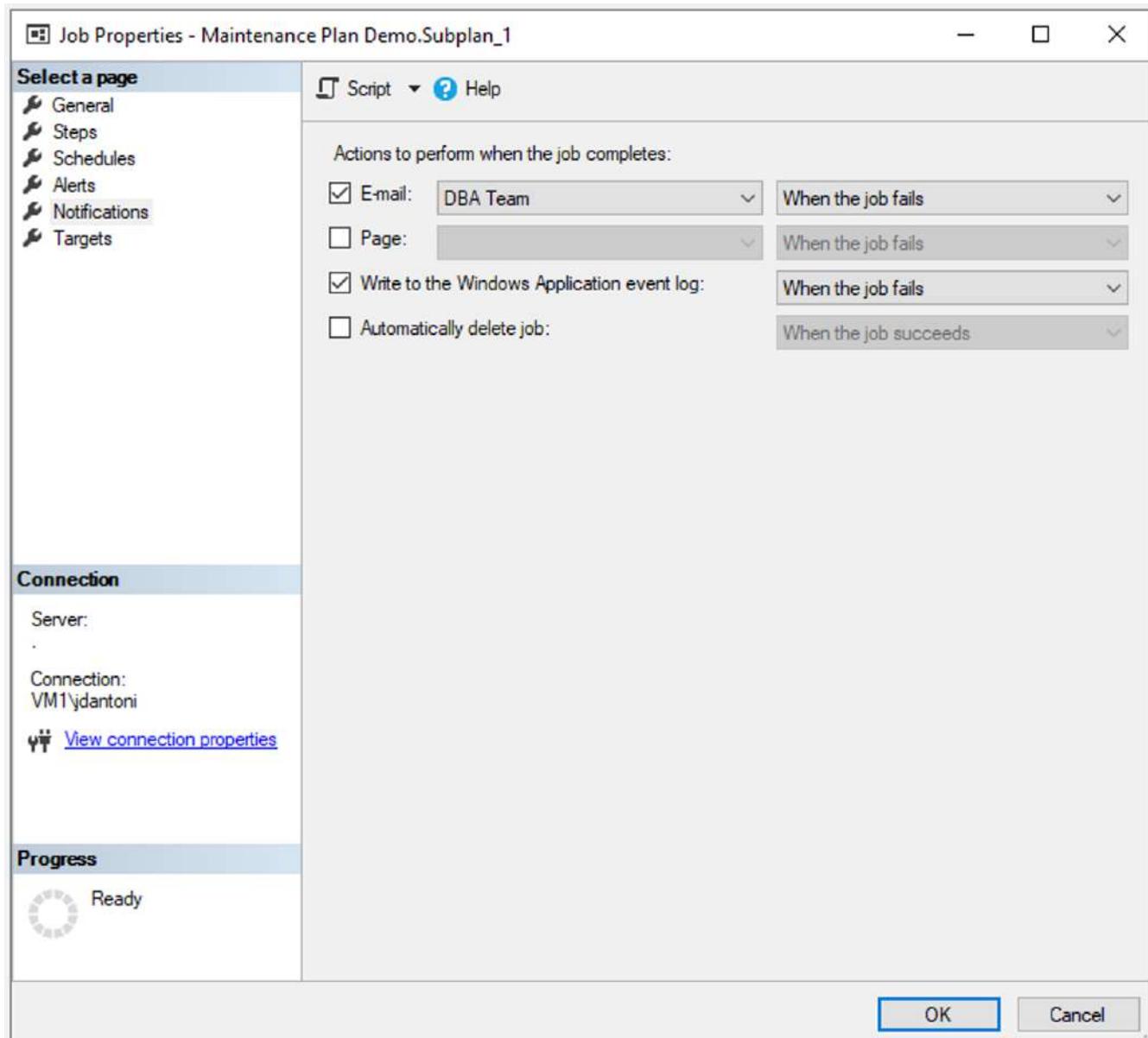
Operators act as an alias for a user or group of users that have been configured to receive notifications of job completion, or to be informed of alerts have been sent to the error log. An operator is defined as an operator name and contact information. Typically, an operator will map to a group of people using an email group. Having multiple people in the email group provides redundancy so that a notification is not missed if someone is unavailable. Groups are also beneficial if an employee leaves the organization; the single person can be removed from the email group and you do not have to update all of your instances. To send email to an operator, you need to enable the email profile of the SQL Server Agent as shown below:



Notifications

Notification of completion is part of each SQL Server Agent job. You have the option of sending a notification on Job completion, failure, or success. Most DBAs notify on failure only, to avoid an influx of notifications for successful jobs. Notifications have a dependency on an operator existing in order to send a notification as shown below:

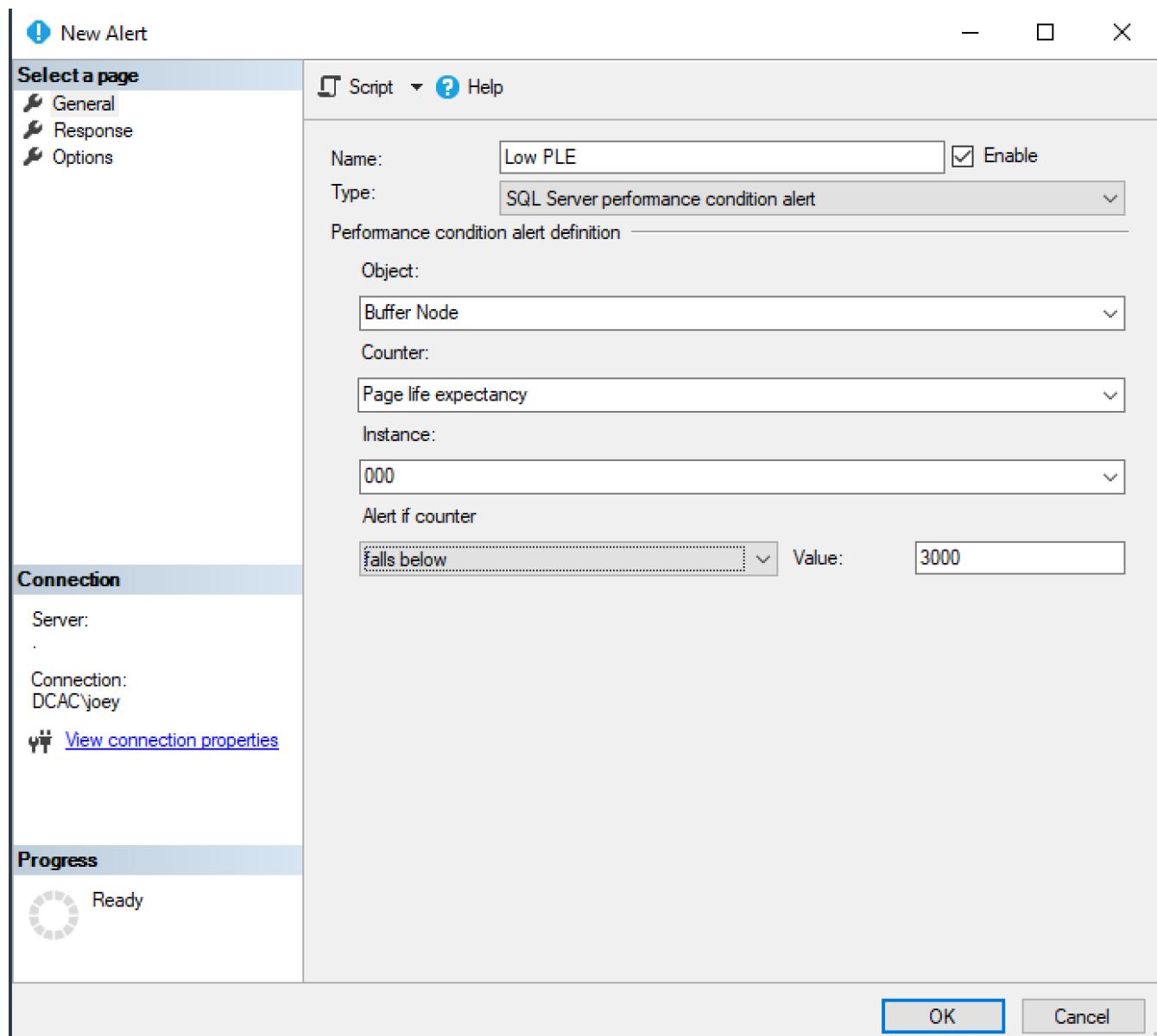
This document belongs to srinivas Ramchand Jillella.
No unauthorized copies allowed!



Alerts

SQL Server Agent alerts allow you to be proactive with monitoring of your SQL Server. The agent reads the SQL Server error log and when it finds an error number for which an alert has been defined, it notifies an operator. In addition to monitoring the SQL Server error log, you can set up alerts to monitor SQL Server Performance conditions, as well as Windows Management Instrumentation (WMI) events. You can specify an alert to be raised in response to one or more events. A common pattern is to raise an alert on all SQL Server errors of level 16 and higher, and then add alerts for specific event types related to critical storage errors or Availability Group failover. Another example would be to alert on performance conditions such as high CPU utilization or low Page Life Expectancy.

Another common use case for alerts is that DBAs may want to be notified in the event of certain server conditions. For example, if CPU utilization is over 90% for a period of five minutes, or Page Life Expectancy drops below a certain value. This is accomplished by creating performance condition alerts. These conditions are based on the Windows Performance Monitor (perfmon) metrics that are tracked within the SQL Server database engine. You can reach the screen shown below by right-clicking **SQL Server Agent** (if it is running) and choosing **New|Alert**.



You have options for how to respond to the performance condition — you can notify an operator via email, which is the most common approach, or you can execute another SQL Server Agent job, which could resolve the problem. Executing another SQL Server Agent job is most commonly used in the scenario where the condition is well-known, and easily handled without manual intervention. A good example of this would be to create an alert on for SQL Server storage error conditions (errors 823, 824, 825), and then to execute a job to perform a database consistency check. The notifications for these alerts use the same SQL Server Agent subsystem.

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What has to be configured before the SQL Server Agent can send email?

- A mail profile
- An agent job
- An alert

Check Answers

Multiple choice

Which system database stores SQL Server Agent jobs and their information?

- Msdb
- Master
- Model

Check Answers

Multiple choice

Which operation recalculates the statistics on an index?

- Rebuild
- Reorganize
- Shrinking a file group

Check Answers

Multiple choice

What built-in option removes old files related to maintenance plans?

- Cleanup history task
- Maintenance cleanup task
- Execute SQL Server agent job task

Check Answers

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Summary

The SQL Server Agent provides a robust mechanism for managing and maintaining your SQL Server and Azure SQL managed instance deployments. In addition to providing job scheduling, it provides alerting and some monitoring for your databases. Maintenance plans can be used to provide backups, index and statistics maintenance, and log management activity. Note that the SQL Server Agent is only available on SQL Server and Azure SQL Managed Instance, but not Azure SQL Database.

Now that you've reviewed this module, you should be able to:

- Describe what maintenance activities you should perform on your databases
- Describe how to configure notifications and alerts on SQL Server Agent jobs and SQL Server
- Describe how to configure notifications alerts based on performance monitor values

Introduction

You've learned about some of the capabilities of the SQL Server Agent. SQL Server Agent is also available on Azure SQL Managed Instance. However, if you're using Azure SQL Database, you'll need an alternative scheduling mechanism, as both the msdb database and the SQL Server agent are unavailable.

In this module you'll learn about Azure Automation, Logic Apps, and elastic jobs, three approaches for automating jobs in PaaS. You'll also learn about Azure policy and how it can be used to manage your subscription and costs.

Learning objectives

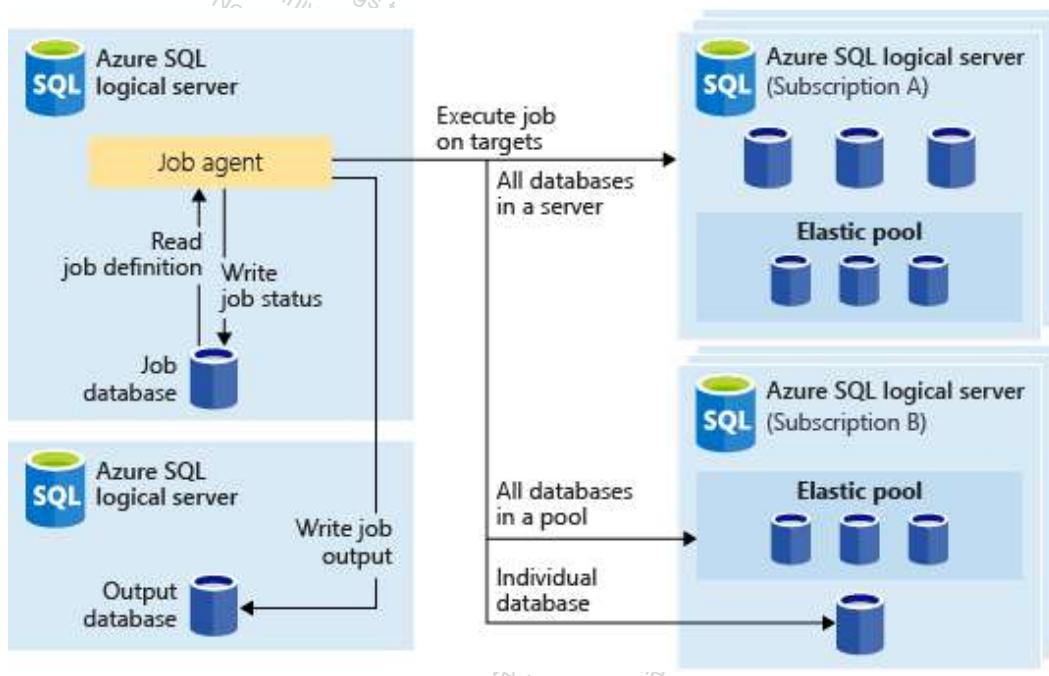
At the end of this module, you'll understand:

- The benefits of Azure policy
- The capabilities of Azure Automation
- How to use elastic jobs
- How to use Logic Apps

Explore Elastic jobs

One of the reasons why many DBAs became so familiar with Azure Automation is that Azure SQL Database initially lacked capabilities for scheduled jobs.

Elastic jobs feature allows you to run a set of T-SQL scripts against a collection of servers or databases as a one-time job, or by using a defined schedule. Elastic jobs work similarly to SQL Server Agent jobs, except that they're limited to executing T-SQL. The jobs work across all tiers of Azure SQL Database. SQL Agent jobs continue to be used for task automation in SQL Server and are also included with Azure SQL Managed Instances.



To configure Elastic Jobs, you need a Job agent and database dedicated to managing your jobs. The recommended service tier for the job database is S1 or higher, and the optimum service tier will be dependent on the number of jobs you're executing and the frequency of those jobs.

Let's review the elastic jobs components:

- **Elastic Job agent** - your Azure resource for running and managing jobs.
- **Job database** - a database dedicated to manage your jobs.
- **Target group** - a collection of servers, elastic pools, and single databases in which a job will be run.
- **Job** - one or more T-SQL scripts that compose a job step.

If a server or elastic pool is the target, a credential within the master database of the server or pool should be created so that the job agent can enumerate the databases within. For a single database, a database credential is all that is needed. Credentials should have the least privileges necessary to perform the job step.

[Home](#) > [Elastic Job agents](#) >
Elastic Job agent ...

Microsoft

[Basics](#) [Review + create](#)

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 *

ElasticJobAgent

Subscription *

Contoso Subscription

Job database

JobSQLDatabase (contoso-east-jp, eastus)

[Select Job database](#)[Review + create](#)[Next : Review + create >](#)

You can create an elastic job agent through the Azure portal. On the **Elastic Job agent** page, make sure you provide a name for your agent, and specify a SQL database for your job database.

You can create a target group by using either PowerShell or T-SQL. The following snippet creates **MyServerGroup** target group including all databases that exist on the server at the time of execution. This code snipped assumes that the variable `$jobAgent` and the variable `$targetServerName` were previously provided.

```
# create MyServerGroup target group
$serverGroup = $jobAgent | New-AzSqlElasticJobTargetGroup -Name 'MyServerGroup'

$serverGroup | Add-AzSqlElasticJobTarget -ServerName $targetServerName -
RefreshCredentialName $masterCred.CredentialName
```

The code snipped below creates an elastic job, and add job steps using PowerShell. The **Step1** is responsible to create the **MyTable** table if it does exist.

```
write-Output "Creating a new job..."
$jobName = "MyFirstElasticJob"
$job = $jobAgent | New-AzSqlElasticJob -Name $jobName -RunOnce

write-Output "Creating job steps for $($jobName) job..."
$sqlText1 = "IF NOT EXISTS (SELECT * FROM sys.tables WHERE object_id = object_id('MyTable'))
CREATE TABLE [dbo].<a href=[Id] [int] NOT NULL" title="" target="_blank" data-
generated='>MyTable</a>;"
```

```
$job | Add-AzSqlElasticJobStep -Name "Step1" -TargetGroupName $serverGroup.TargetGroupName -  
CredentialName $jobCred.CredentialName -CommandText $sqlText1
```

As we can see above, T-SQL scripts being executed by elastic jobs should be idempotent, which means if the job is run multiple times, whether accidentally or because of job failure, the job won't fail or produce unintended results. You should be able to run the same script multiple times without failure.

Finally, run the elastic job *MyFirstElasticJob* using PowerShell.

```
write-Output "Start the job..."  
$jobExecution = $job | Start-AzSqlElasticJob  
$jobExecution
```

Use case scenarios

Elastic jobs can be used in the following scenarios:

- Automate management tasks to run on a specific schedule
- Deploy schema changes
- Data movements
- Collect, and aggregate data for reporting or other purposes
- Load data from Azure Blob storage
- Configure jobs to execute across a collection of databases on a recurring basis, such as during off-peak hours
- Data processing over a large number of databases, for instance, telemetry collection. Results are then compiled into a single destination table for further analysis.

Understand Azure Automation

Azure offers several ways to automate processes. Azure Functions and Logic Apps are both Azure services that enable serverless workloads. Both services create workflows that are a collection of steps to execute complex tasks. For example, a Logic App can be created to populate a table in an Azure SQL Database when an entry is made in a SharePoint list. A full explanation of these services is beyond the scope of this course.

For more complete control and granularity of your automation, Azure Automation allows for process automation, configuration management, full integration with Azure platform options (such as role-based access control and Azure AD) and can manage Azure and on-premises resources.

One of the unique benefits of Azure Automation is that it can manage resources within Azure or on-premises VMs. For example, if you have a VM that is normally kept in a down state for cost savings (except when it needs to be used), you have the ability within Azure Automation, using a feature called hybrid runbooks, to execute a script to start the VM, then kick off a SQL Server backup from within the VM, and finally shut down the VM.

Another common scenario is to use Azure Automation for periodic maintenance operations, such as purging stale or old data, or reindex a SQL database.

Azure Automation components

Azure Automation supports both automation and configuration management activities. We're going to focus on the automation components, but Azure Automation can also be used to manage server updates and desired state configuration. The components of Azure Automation you'll need to use to execute automated tasks are as follows:

- **Runbooks** - Runbooks are the unit of execution in Azure Automation. Runbooks can be defined as one of three types: a graphical runbook based on PowerShell, a PowerShell script, or Python script. PowerShell runbooks are most commonly used to manage Azure SQL resources.
- **Modules** - Azure Automation defines an execution context for the PowerShell or Python code you're executing in your runbook. In order to execute your code, you need to import the supporting modules. For example, if you needed to run the `Get-AzSqlDatabase` PowerShell cmdlet, you would need to import the Az.SQL PowerShell module into your automation account.
- **Credentials** - Credentials store sensitive information that runbooks or configurations can use at runtime.
- **Schedules** - Schedules are linked to runbooks and trigger a runbook at a specific time.

Azure policy

Group Policies, or GPOs, have been used by Windows server administrators for a long time, to manage security, provide consistency across the Windows Server environment in your organization. Some examples of group policies are enforcement of password complexity, mapping shared network drives and configuring networked printers.

Azure Policy includes initiative definitions to help establish and maintain compliance with different security standards for your Automation account. Similar features are also available in Azure Resource Manager. A policy provides a level of governance over your Azure subscriptions. Policy can enforce rules and controls over your Azure resources. Some examples of how you might use this include limiting the regions you can deploy a resource to, enforcing naming standards, or controlling resource sizes. Azure provides many example policies that you can use or you can define custom policies using JSON.

Policies are assigned to a specific scope, which could be a management group (a group of subscriptions that are managed together), a subscription, a resource group, or even an individual resource. Most commonly policy will be applied at the subscription or resource group level. Individual policies can be grouped using a structure known as initiatives, which are sometimes called policy sets. Policies have a scope of assignment that can be defined at the individual resource, the resource group, the subscription, or a management group (a group of subscriptions managed together), or all of the subscriptions in a given tenant.

Another example of how you might implement Azure Policy is tagging of resources. Azure tags, which are described below, store metadata about Azure resources in key-value pairs, and are commonly used to highlight environment type (test, QA, or production) or cost center for a given resource. A policy that required all resources to have a tag for environment and cost center would cause an error and block the deployment of any Azure resource that didn't have the required tags.

Azure subscriptions and tags

Organizations use multiple subscriptions for several reasons, including budget management, security, or isolation of resources. One example of this would be an organization that has both internal and customer facing resources. The internal resources could exist in one subscription, and the customer resources in another, for easier separation of billing and for isolation of the internal resources. These subscriptions may be managed together in a management group, which allows you to manage policy and compliance across subscriptions.

Tags are simply metadata that are used to better describe your Azure resources. These tags are stored as **key:value** pairs and appear in the Azure portal associated with your Azure resources. Since they're associated with the resource, when you use PowerShell or Azure CLI commands, you can filter your commands based on tags. In that sense, you can think of them like a **WHERE** clause in a SQL query. A basic example is shown below:

```
$rg=(get-AzResourceGroup)

$rg=($rg|where-object {($_.tags['Use'] -ne 'Internal')}).ResourceGroupName
```

On the second line of this code sample, you can see that the list of resource groups is being filtered by the tag called '*Use*', and will return only those resource groups where that tag doesn't have a value of '*Internal*'. Tags can be applied in the Azure portal, or programmatically via PowerShell, Azure CLI, or as part of your deployment process. Tags can also be applied at the subscription, resource group, or individual resource level. Tags can also be modified at any time. Azure supports applying up to 15 tags to each Azure resource.

Tags are also included in Azure billing information, so tagging by cost center means it can be much easier for management to break down the Azure charges. Tags are in the overview section of the blade for every Azure Resource. To add tags to a resource using the Azure portal, select tags, and enter the key and value for your tag. Select save after you apply the tags to your resources.

Name	Value
Cost Center	: Finance
Environment	: Development

dp300-lab06-xyz/AdventureWor... (SQL database)
Environment : Development Cost Center : Finance
No changes

You can also use PowerShell or the CLI to add tags. The PowerShell example is below:

```
$tags = @{"Dept"="Finance"; "Status"="Normal"}  
  
$resource = Get-AzResource -Name demoStorage -ResourceGroup demoGroup  
  
New-AzTag -ResourceId $resource.id -Tag $tags
```

The Azure CLI example is below:

```
az resource tag --tags 'Dept=IT' 'Environment=Test' -g examplegroup -n examplevnet `  
--resource-type "Microsoft.Network/virtualNetworks"
```

Tags enable customers to organize Azure resources and management hierarchy into a taxonomy.

NOTE: Tags are stored as plain text. Never add sensitive values to tags. Sensitive values could be exposed through many methods, including cost reports, tag taxonomies, deployment histories, exported templates, and monitoring logs.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Build an automation runbook

As we've explored how Azure Automation works, we'll look at the steps required to create an automation account and an automation runbook.

In order to build an automation runbook, you need to first create an automation account. The image below shows this process in the Azure portal, after selecting Azure Automation from the Azure Marketplace.

The screenshot shows the 'Create an Automation Account' wizard in the Azure portal. The top navigation bar includes 'Home > Marketplace > Automation > Create an Automation Account ...'. The main tabs are 'Basics', 'Advanced', 'Networking', 'Tags', and 'Review + Create'. The 'Basics' tab is selected. The 'Subscription' dropdown is set to 'Contoso Subscription'. The 'Resource group' dropdown is set to 'myresourcegroup' with a 'Create new' link below it. Under 'Instance Details', the 'Automation account name' is 'MyAutomationAcct' and the 'Region' is 'West US'. At the bottom, there are 'Review + Create', 'Previous', and 'Next' buttons.

In this example runbook, you're going to connect to an Azure SQL Database using PowerShell. This means you need to import modules to support those cmdlets. Before you create your runbook, you'll import modules into your Azure Automation account. In order to do this import, navigate to the Shared Resources section of the main blade for your automation account and select **Modules Gallery**. The first module you'll import, is **Az.Accounts** as the **Az.SQL** module is dependent on it.

A screenshot of the Azure PowerShell - Accounts module page in the gallery. The search bar at the top contains the text "az.accounts". The module card for "Az.Accounts" is displayed, showing its description: "Microsoft Azure PowerShell - Accounts credential management cmdlets for Azure Resource Manager in Windows PowerShell and PowerShell Core.", its tags: "Azure ResourceManager ARM Accounts Authentication Environment Subscription PSModule PSEdition_Core PSEdition/Desktop", and its metadata: "Created by: azure-sdk", "11540595 downloads", and "Last updated: 4/16/2020". A watermark "This document belongs to srinivas Ramchand Jillela. No unauthorized copies allowed!" is visible across the page.

Search for the module in the gallery as shown in the image above. After you select the module, select **Import**, as shown in the image below.

A screenshot of the "Az.Accounts" module import dialog. It shows the module's description: "Microsoft Azure PowerShell - Accounts credential management cmdlets for Azure Resource Manager in Windows PowerShell and PowerShell Core.", and a link to more information: "For more information on account credential management, please visit the following: <https://docs.microsoft.com/powershell/azure/authenticate-azureps>". A large blue "Import" button is prominently displayed. A watermark "This document belongs to srinivas Ramchand Jillela. No unauthorized copies allowed!" is visible across the page.

This will import the module into your account. In this example, the process was repeated for the Az.SQL and SqlServer PowerShell modules.

Next, you can optionally create a credential that your runbook can use. You can create a credential by clicking on **Credentials** in the **Shared Resources** section of the main blade of your automation account as shown in the image below. You don't have to create a credential in order to use Azure Automation, but this example does refer to one.



New Credential X

Name *

 ✓

Description

User name *

 ✓

Password *

 ✓

Confirm password *

 ✓

Create

On the **Process Automation** section of your Automation Account, select **Runbooks**, to create a runbook. Your account will come with two sample runbooks.

[Home](#) > [Microsoft.AutomationAccount](#) > [MyAutomationAcc](#) >

Create a runbook

Name * ⓘ DemoDP300 ✓

Runbook type * ⓘ PowerShell ▾

Runtime version * ⓘ 7.1 (preview) ▾

Description

i During runbook execution, PowerShell modules targeting 7.1 runtime version will be used. Please make sure the required PowerShell modules are present in 7.1 runtime version.

[Create](#)[Cancel](#)

To create a runbook, you must provide a name, the type of runbook, the runtime version, and optionally a description. Because this example specified PowerShell as the type, a PowerShell editor opens.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belo...

```

1 # Ensures you do not inherit an AzContext in your runbook
2 Disable-AzContextAutosave -Scope Process
3
4 $connection = Get-AutomationConnection -Name AzureRunAsConnection
5
6 # Wrap authentication in retry logic for transient network failures
7 $logonAttempt = 0
8 while(!$connectionResult) -And ($logonAttempt -le 10)
9 {
10     $logonAttempt++
11     # Logging in to Azure...
12     $connectionResult = Connect-AzAccount `-
13         -ServicePrincipal `-
14         -Tenant $connection.TenantID `-
15         -ApplicationId $connection.ApplicationID `-
16         -CertificateThumbprint $connection.CertificateThumbprint
17
18     Start-Sleep -Seconds 30
19 }
20
21 $AzureContext = Get-AzSubscription -SubscriptionId $connection.SubscriptionID
22
23 $dbname=(Get-AzSQLDatabase -ResourceGroupName 'SQLDB' -ServerName 'GSData' -Database 'GSData').DatabaseName
24
25 $AzureSQLServerName = $dbname + ".database.windows.net"
26 $Cred = Get-AutomationPSCredential -Name "SQLUser"
27 $SQLOutput = $(Invoke-Sqlcmd -ServerInstance $AzureSQLServerName -Username $Cred.UserName -Password $Cred.GetNetworkCredential().Password `-
28 -Database $DBName -Query "SELECT * FROM INFORMATION_SCHEMA.TABLES" -Verbose) >&1
29
30 Write-Output $SQLOutput
31
32
33
34

```

The image above shows the edit runbook screen, which is where you define the code you're executing. In this example, the runbook is connecting to the Azure subscription, getting information about an Azure SQL Database, running a query, and then returning the results.

In the lines 1-21, you're executing a series of **cmdlets** to connect to the Azure account. You're then getting the database name from the **Get-AzSQLDatabase** cmdlet, and then using the **get-AutomationPSCredential** cmdlet to assign your credential to a variable.

Finally, you're assigning the **invoke-sqlcmd** cmdlet to execute a query against the Azure SQL Database, and using the **write-output** cmdlet to return the results of the query.

After you've completed your code in the portal, select **Test pane** in the code editor in the Azure portal. This allows you to test your code in the context of Azure Automation. A typical development process is to create your PowerShell code locally, and then test it within the automation environment. This allows you to separate any PowerShell errors from errors that might be generated from the context of automation execution. Always test your code within automation, to ensure there are no errors in the code itself.

```

2 Disable-AzContextAutosave -Scope Process
3
4 $connection = Get-AutomationConnection -Name AzureRunAsConnection
5
6 # Wrap authentication in retry logic for transient network failures
7 $logonAttempt = 0
8 while(!$connectionResult -And ($logonAttempt -le 10))
9 {
10     $logonAttempt++
11     # Logging in to Azure...
12     $connectionResult = Connect-AzAccount ` 
13         -ServicePrincipal ` 
14         -Tenant $connection.TenantID ` 
15         -ApplicationId $connection.ApplicationID ` 
16         -CertificateThumbprint $connection.CertificateThumbprint
17
18     Start-Sleep -Seconds 30
19 }
20
21 $AzureContext = Get-AzSubscription -SubscriptionId $connection.SubscriptionID
22
23 $dbname=(Get-AzSQLDatabase -ResourceGroupName 'SQLDB' -ServerName 'GSData' -Database 'GSData').DatabaseName
24
25 $AzureSQLServerName = $dbname + ".database.windows.net"
26 $Cred = Get-AutomationPSCredential -Name "SQLUser"
27 $SQLOutput = $(Invoke-Sqlcmd -ServerInstance $AzureSQLServerName -Username $Cred.UserName -Password $Cred.GetNetworkCredential().Password -Database $DBName -Query "SELECT * FROM INFORMATION_SCHEMA.TABLES" -Verbose) 4>&1
28
29 Write-Output $SQLOutput
30
31
32
33
34

```

The image below shows the results of the completed runbook. Note the informational bubble on the left side of the screen referring to hybrid runbooks. Hybrid runbooks are used when you need to execute cmdlets inside of a virtual machine. You'll need a configuration on the virtual machines and in the Azure Automation account. This concept can be a bit confusing, but the easiest way to think about it's to think of Azure resources as boxes that are managed by Azure Resource Manager. Without a hybrid runbook you can manage the state of those boxes, but you can't access or manage anything within the boxes. Hybrid runbooks give you the option to control what's inside of the box.

*This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!*

Test
DemoDP300

► Start Stop Suspend ↻ Resume ⏱ View last test ⌂ Refresh job streams

Parameters
No input parameters

Run Settings
Run on Azure

Activity-level tracing
This configuration is available only for graphical runbooks.

Trace level
None Basic Detailed

Completed

```
Mode : Process
ContextDirectory :
ContextFile :
CacheDirectory :
CacheFile :
Settings : {}



| TABLE_CATALOG | TABLE_SCHEMA | TABLE_NAME              | TABLE_TYPE |
|---------------|--------------|-------------------------|------------|
| gsdata        | sys          | database_firewall_rules | VIEW       |


```

After you've successfully tested your runbook, you can then select **Publish** in the runbook editor screen.

A runbook must be published in order to be executed by the Azure service. After you've published the runbook, you can create a schedule by selecting **Schedules** in the **Shared Resources** section of the automation account blade.

The image above shows the creation process for a new schedule. The default settings are for there to be no recurrence of the job. In the above example, the job has been configured to run once daily at 4:00 PM Central Time.

Once you've created a schedule, you can link it to a runbook by navigating back to the runbook, and selecting **Link to schedule** in the runbook page as shown in the image below.

This document belongs to srinivas Ramchand Jillella.
No unauthorized copies allowed!

Automate database workflows by using Logic Apps

Azure Logic Apps is a cloud-based platform for creating and running automated workflows that integrate your apps, data, services, and systems. With this platform, you can quickly develop highly scalable integration solutions for your enterprise and business-to-business (B2B) scenarios. As a member of Azure Integration Services, Azure Logic Apps simplifies the way that you connect legacy, modern, and cutting-edge systems across cloud, on premises, and hybrid environments.

The following list describes just a few example tasks, business processes, and workloads that you can automate using the Azure Logic Apps service:

- Schedule and send email notifications using Office 365 when a specific event happens, for example, a new file is uploaded.
- Route and process customer orders across on-premises systems and cloud services.
- Move uploaded files from an SFTP or FTP server to Azure Storage.
- Monitor tweets, analyze the sentiment, and create alerts or tasks for items that need review.

Why use Azure Logic Apps?

The Azure Logic Apps integration platform provides prebuilt Microsoft-managed API connectors and built-in operations so you can connect and integrate apps, data, services, and systems more easily and quickly, where you can focus on designing and implementing your solution's business logic and functionality, not on figuring out how to access your resources.

You usually won't have to write any code. However, if you do need to write code, you can create code snippets using Azure Functions and run that code from your workflow. You can also create code snippets that run in your workflow by using the Inline Code action. If your workflow needs to interact with events from Azure services, custom apps, or other solutions, you can monitor, route, and publish events using Azure Event Grid.

Logic Apps is fully managed by Microsoft Azure, which frees you from worrying about hosting, scaling, managing, monitoring, and maintaining solutions built with these services. When you use these capabilities to create "serverless" apps and solutions, you can just focus on the business logic and functionality. These services automatically scale to meet your needs, make integrations faster, and help you build robust cloud apps using little to no code.

SQL Server connector

The SQL Server connector allows you to access your SQL database with the SQL Server connector in Azure Logic Apps. You can then create automated workflows that are triggered by events in your SQL database or other systems and manage your SQL data and resources.

For example, you can use actions that get, insert, and delete data along with running SQL queries and stored procedures. You can create workflow that checks for new records in a non-SQL database, does some processing work, creates new records in your SQL database using the results, and sends email alerts about the new records in your SQL database.

The SQL Server connector supports the following SQL editions:

- SQL Server
- Azure SQL Database
- Azure SQL Managed Instance

The SQL Server connector requires that your tables contain data so that SQL connector operations can return results when called. For example, if you use Azure SQL Database, you can use the included sample databases to try the SQL connector operations.

For a SQL database in Azure, the connection string has the following format:

```
Server=tcp:{server-name}.database.windows.net,1433;Initial Catalog={database-name};P
```

Alternatively, you can also check the connection string for your Azure SQL Database in the Azure portal. On the **Overview** section for your database, select **Show database connection strings** for **Connection strings** property.

If you want to start your workflow with a SQL Server trigger operation, you have to start with a blank workflow.

The SQL Server connector is available for logic app workflows in multi-tenant Azure Logic Apps, integration service environment (ISE), and single-tenant Azure Logic Apps:

- **Consumption workflows in multi-tenant Azure Logic Apps** – this connector is available only as a managed connector. For more information, review the [managed SQL Server connector operations](#).
- **Consumption workflows in an integration service environment** – this connector is available as a managed connector and as an ISE connector that's designed to run in an ISE. For more information, review the [managed SQL Server connector operations](#).
- **Standard workflows in single-tenant Azure Logic Apps** – this connector is available as a managed connector and as a built-in connector that's designed to run in the same process as the single-tenant Azure Logic Apps runtime. However, the built-in version differs in the following ways:
 - The built-in SQL Server connector has no triggers.
 - The built-in SQL Server connector has only one operation: Execute Query

Create a logic app workflow

The following steps use the Azure portal to create logic app workflows:

Add a SQL Server trigger

The following steps use the Azure portal, but with the appropriate Azure Logic Apps extension, you can also use Visual Studio Code to create logic app workflows:

1. In the Azure portal, open your blank logic app workflow in the designer.
2. Find and select the managed SQL Server connector trigger that you want to use. Under the designer search box, select **All**.
3. In the designer search box, enter *sql server*.
4. From the triggers list, select the SQL trigger that you want. This example uses the trigger named **When an item is created**.

The screenshot shows the Azure Logic App designer interface. On the left, a sidebar lists various options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Development Tools, Logic app designer (which is selected and highlighted in grey), Logic app code view, Versions, API connections, and Quick start guides. The main area has a search bar at the top with 'sql server' typed in. Below it, tabs for 'For You', 'All', 'Built-in', 'Standard', 'Enterprise', and 'Custom' are visible. Under the 'All' tab, three items are listed: 'Cloudmiserive Security' (with a 'C' icon), 'SQL Server' (with a monitor icon), and 'SQL Server Analysis...' (with a cube icon). A dropdown menu is open over the 'SQL Server' item. Below this, two triggers are shown under the 'Triggers' tab: 'When an item is created (V2) SQL Server' and 'When an item is modified (V2) SQL Server'. The first trigger is highlighted with a red border.

5. If you're connecting to your SQL database for the first time, you're prompted to create your SQL database connection now. After you create this connection, you can continue with the next step.
6. In the trigger properties, specify the interval and frequency for how often the trigger checks the table.
7. To add other properties available for this trigger, open the **Add new parameter** list and select those properties.
NOTE: This trigger returns only one row from the selected table, and nothing else. To perform other tasks, continue by adding either a SQL Server connector action or another action that performs the next task that you want in your logic app workflow.
For example, to view the data in this row, you can add other actions that create a file that includes the fields from the returned row, and then send email alerts. To learn about other available actions for this connector, see the connector's reference page.
8. On the designer toolbar, select **Save**. This step automatically enables and publishes your logic app live in Azure.

Add a SQL Server action

The following steps use the Azure portal. In this example, the logic app workflow starts with the Recurrence trigger, and calls an action that gets a row from a SQL database.

1. In the Azure portal, open your logic app workflow in the designer.
2. Find and select the managed SQL Server connector action that you want to use. This example uses the action named **Get row**.
3. Under the trigger or action where you want to add the SQL action, select **New step**.
4. In the **Choose an operation** box, under the designer search box, select **All**.
5. In the designer search box, enter *sql server*.
6. From the actions list, select the SQL Server action that you want. This example uses the **Get row** action, which gets a single record.

ConsumptionLogicApp | Logic app designer

Search (Ctrl+ /) Save Discard Run Trigger Designer Code view Parameters

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Development Tools
- Logic app designer
- Logic app code view
- Versions
- API connections
- Quick start guides
- Settings
- Workflow settings
- Authorization
- Access keys
- Identity
- Properties
- Locks
- Monitoring

Recurrence

Choose an operation

sql server

For You All Built-in Standard Enterprise Custom

SQL Server Cloudmersive Security SQL Server Analysis...

Triggers Actions

C Automatically detect threats in an input string (preview)
Cloudmersive Security

SQL Server Delete row (V2)

SQL Server Execute a SQL query (V2)

SQL Server Execute stored procedure (V2)

SQL Server Get row (V2)

7. If you haven't already provided the SQL server name and database name, provide those values. Otherwise, from the Table name list, select the table that you want to use. In the Row ID property, enter the ID for the record that you want. In this example, the table name is *SalesLT.Product*.

Recurrence

Get row (V2)

* Table name: SalesLT.Product

* Row id: Unique identifier of the row to retrieve

Connected to MySQLServerConnection. Change connection.

+ New step

NOTE: This action returns only one row from the selected table, and nothing else.

8. When you're done, on the designer toolbar, select **Save**.

Connect to Azure SQL Database

In the workflow designer, you must create a connection the first time you add a trigger or action for the first time. This information varies depending on the connection, for example:

- The name that you want to use for the new connection
- The name for the system or server
- Your user or account credentials
- The authentication type to use

Monitor automated tasks

After you automate your tasks, it's important to monitor them to make sure they're working properly. That way you can maximize performance and availability of your services and proactively identify problems.

Monitor runbooks

Your runbooks should be modular in nature, with logic that can be reused and restarted easily. Monitoring progress in a runbook ensures that the logic of the runbook is executed correctly if issues arise.

A database, storage account, or shared file can be used to monitor a runbook's progress. Ensure that your runbook checks the last action performed before initiating the next action. Based on the results of the check, the logic can either skip or continue specific tasks in the runbook.

You can monitor runbooks execution in Azure portal. Select **Job** on the **Process Automation** section of the main blade of your automation account.

Runbook	Job created	Status	Ran on	Last status update
DevOp300	4/26/2022, 1:21:11 PM	✓ Completed	Azure	4/26/2022, 1:21:21 PM
AzureAutomationTutorialWithIdentity	4/26/2022, 1:16:21 PM	✓ Completed	Azure	4/26/2022, 1:16:45 PM

As you can see in the image above, the runbook highlighted was completed. To investigate the details and status, select the job.

To learn more about how to troubleshoot runbooks problems, see [Troubleshoot runbook issues](#)

Alerts

You can also create metric alerts to monitor the execution of your runbooks. Alerts allow you to define conditions to monitor and actions to take when those conditions are met.

The screenshot shows the Azure portal interface for managing an Automation Account named 'MyAutomationAcc'. On the left, a navigation menu includes 'Linked workspace', 'Event grid', 'Start/Stop VM', 'Account Settings' (with options like Properties, Networking, Keys, Pricing, Source control, Run as accounts, and Identity), 'Settings' (Locks), and 'Monitoring' (Alerts, Metrics, Diagnostic settings, Logs). The 'Alerts' option under Monitoring is highlighted with a red box. In the main content area, there's a message: 'Set up alert rules on this resource. Get notified when important monitoring events happen on your resource.' Below this is a 'Create alert rule' button, which is also highlighted with a red box. A large green exclamation mark icon is visible on the right.

When you select **Create alert rule**, the **Select a signal** slide out will open on the right side of your screen. Next, you'll need to select a signal that is most appropriate for your scenario. For this example, select **Total Jobs**.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Select a signal

X

Choose a signal below and configure the logic on the next screen to define the alert condition.

Signal type ⓘ

All

Monitor service ⓘ

All

Displaying 1 - 10 signals out of total 10 signals

Signal name	↑↓	Signal type	↑↓	Monitor service	↑↓
Custom log search	Log search	Log analytics			
Total Jobs	Metrics	Platform			
Total Update Deployment Machine Runs	Metrics	Platform			
Total Update Deployment Runs	Metrics	Platform			
All Administrative operations	Activity log	Administrative			
Convert Graph Runbook Content to its raw serialized format and vice-versa (Micr...)	Activity log	Administrative			
Generate a URI for an Azure Automation webhook (Microsoft.Automation/automat...)	Activity log	Administrative			
Create or Update an Azure Automation account (Microsoft.Automation/automati...)	Activity log	Administrative			
Gets the Keys for the automation account (Microsoft.Automation/automationAcc...)	Activity log	Administrative			
Delete an Azure Automation account (Microsoft.Automation/automationAccounts)	Activity log	Administrative			

Done

In the **Configure signal logic** slide out, select **Static** for the **Threshold** property. Then, set the **Operator** property to **Greater than**, and the **Aggregation** type to **Total**. Then in **Threshold value** enter a value of **10**.

This document belongs to
asg.dba@gmail.com
~ unauthenticated users allowed!
srinivas.lamantala.jillella.

Configure signal logic

X



Split by dimensions

Use dimensions to monitor specific time series and provide context to the fired alert. Dimensions can be either number or string columns. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. [About monitoring multiple time series](#)

Dimension name	Operator	Dimension values
Select dimension	=	0 selected
<input type="button" value="Add custom value"/>		

Alert logic

i Monitoring 1 time series (\$0.1/time series)

Threshold (1)

Operator (1)	Aggregation type * (1)	Threshold value * (1)	Unit * (1)
Greater than	Total	10	Count

Condition preview

Whenever the total total jobs is greater than 10

Evaluated based on

Aggregation granularity (Period) * (1)	Frequency of evaluation (1)
5 minutes	Every 5 Minutes

Done

Alternatively, you can specify a dimension. For example, you can define that an alert rule will only trigger for a specific runbook and **status**. If you don't specify a dimension, then no filter will be applied.

Split by dimensions

Use dimensions to monitor specific time series and provide context to the fired alert. Dimensions can be either number or string columns. If you select more than one dimension value, each time series that results from the combination will trigger its own alert and will be charged separately. [About monitoring multiple time series](#)

The screenshot shows the Azure portal interface for creating an alert rule. In the main pane, there are two filter conditions: 'Runbook Name' set to 'All current and future values' and 'Status' set to 'All'. An 'Add custom value' button is highlighted with a red box. A modal window titled 'Add custom dimension value' is displayed, asking for a 'Runbook Name' and providing a dropdown menu with 'DemoDP300' selected. The 'OK' button is also highlighted with a red box.

Alert logic

Next, make sure you configure an action group. An action group is a collection of actions that you can use across multiple alerts. Among these are email notifications, runbooks, webhooks, and many others.

NOTE: You can combine Azure Automation with Azure Alerts action groups to initiate an Automation runbook when an alert is raised.

Activity log

In Azure Automation, runbooks are executed and details are collected in an activity log.

The screenshot shows the 'Activity log' section of the Azure portal for an automation account. The left sidebar has a 'MyAutomationAcc' navigation tree. The 'Activity log' tab is selected and highlighted with a red box. The main area displays a table with 12 items, each representing an operation like 'Publish an Azure Automation runbook draft' or 'Create an Azure Automation job'. The columns include Operation name, Status, Time, Time stamp, Subscription, and Event initiated by. The initiator email for most entries is 'contoso@contoso.com'.

Operation name	Status	Time	Time stamp	Subscription	Event initiated by
① Publish an Azure Automation runbook draft	Successed	38 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Create an Azure Automation job	Accepted	38 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Write an Azure Automation runbook draft	Successed	38 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Write an Azure Automation runbook draft	Successed	38 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Publish an Azure Automation runbook draft	Successed	38 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Write an Azure Automation runbook draft	Successed	38 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Publish an Azure Automation runbook draft	Accepted	39 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Create an Azure Automation job	Accepted	43 minutes...	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Create or Update an Azure Automation schedule asset	Successed	2 hours ago	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Create or Update an Azure Automation Runbook	Successed	2 hours ago	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Create or Update an Azure Automation Runbook	Successed	2 hours ago	Tue Apr 26...	Contoso Subscription	contoso@contoso.com
① Write an Azure Automation runbook draft	Successed	2 hours ago	Tue Apr 26...	Contoso Subscription	contoso@contoso.com

In the image above, you can retrieve runbook details, such as the person or account that started a runbook.

As an alternative, the following PowerShell example provides the last user to run the specified runbook.

```
$rgName = 'MyResourceGroup'
$accountName = 'MyAutomationAccount'
$runbookName = 'MyRunbook'
$startTime = (Get-Date).AddDays(-1)
```

```
$params = @{
    ResourceGroupName = $rgName
    StartTime        = $startTime
}
$JobActivityLogs = (Get-AzLog @params).Where( { $_.Authorization.Action -eq 'Microsoft.Automation/automationAccounts/jobs/write' })

$JobInfo = @{}
foreach ($log in $JobActivityLogs) {
    # Get job resource
    $JobResource = Get-AzResource -ResourceId $log.ResourceId

    if ($null -eq $JobInfo[$log.SubmissionTimestamp] -and
$JobResource.Properties.Runbook.Name -eq $runbookName) {
        # Get runbook
        $jobParams = @{
            ResourceGroupName      = $rgName
            AutomationAccountName = $accountName
            Id                   = $JobResource.Properties.JobId
        }
        $Runbook = Get-AzAutomationJob @jobParams | Where-Object RunbookName -EQ
$runbookName

        # Add job information to hashtable
        $JobInfo.Add($log.SubmissionTimestamp, @{$Runbook.RunbookName, $log.Caller,
$JobResource.Properties.jobId})
    }
}
$JobInfo.GetEnumerator() | Sort-Object Key -Descending | Select-Object -First 1
```

Log Analytics

Azure Automation can send runbook job status and job streams to your Log Analytics workspace. This method doesn't require workspace linking and is independent.

Azure Monitor logs integrated with Automation account, enables you to:

- View the status of your Automation jobs
- Write advanced queries across your job workflow
- Trigger an email or alert based on your runbook job status
- Correlate data from multiple Automation jobs

To run queries, select **Logs** from the **Monitoring** section of your automation account main blade.

The Azure portal provides a few query templates so you can get started. As we can see below, we used an existing Kusto query template to list all completed jobs in the automation account.

The screenshot shows the Azure Log Analytics workspace interface. On the left, there's a sidebar with 'Favorites' and a 'Automation Jobs' section where a specific query is highlighted with a red box. The main area displays a table of query results.

TimeGenerated [UTC]	RunbookName_s	ResultType
4/26/2022, 7:24:57.477 P...	DemoDP300	Completed
	TimeGenerated [UTC]	2022-04-26T19:24:57.477Z
	RunbookName_s	DemoDP300
	ResultType	Completed

The ability to forward Azure Automation diagnostic logs to a Log Analytics workspace is an important feature that helps you monitor the health of your runbooks.

NOTE: Before you start using Log Analytics to query Automation jobs data, you must configure **Diagnostic settings** for your Automation Account.

Monitor elastic jobs (preview)

With elastic jobs, job executions are logged, and any failures are automatically retried. The automatic retry feature provides services more resilient to transient failures.

You can monitor elastic jobs execution through Azure portal, PowerShell, and T-SQL.

Azure portal

To view the history of jobs execution, select **Overview** for your Elastic Job agent main blade.

PowerShell

The following code snippets get job execution details.

```
# get the latest 5 executions run
$jobAgent | Get-AzSqlElasticJobExecution -Count 5

# get the job step execution details
$jobExecution | Get-AzSqlElasticJobStepExecution

# get the job target execution details
$jobExecution | Get-AzSqlElasticJobTargetExecution -Count 2
```

T-SQL

The following example shows how to view execution status details for all jobs. Create a job agent and connect to the job database, then run the following command:

```
--View all execution status for all jobs
SELECT *
FROM jobs.job_executions
ORDER BY start_time DESC;

--View all execution statuses for job named 'MyJob'
SELECT * FROM jobs.job_executions
WHERE job_name = 'MyJob'
ORDER BY start_time DESC;

-- View all active executions
SELECT *
FROM jobs.job_executions
WHERE is_active = 1
ORDER BY start_time DESC;
```

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What is the unit of execution for your Azure Automation Account?

- Runbook
- Schedule
- Container

Check Answers

Multiple choice

What scope can Azure Policy be deployed to?

- Tenant
- Subscription
- User

Check Answers

Multiple choice

What is the name for the scope that you must define for a SQL Elastic Job?

- Target Group
- Management Group
- Resource Group

Check Answers

Summary

In this module you learned about various ways to automate common tasks in Azure and within SQL Server. One of the benefits of cloud computing is that a framework for robust automation, monitoring, and tooling is part of the platform.

In the on-premises world, a complex set of tools would need to be assembled just to match the built-in functionality available with Azure metrics, policy, and automation. Additionally, SQL Server provides an automation framework using SQL Agent, and a very robust monitoring solution via Extended Events. You can take advantage of both Azure Automation and Azure elastic jobs to automate the management of your Azure resources.

Now that you've reviewed this module, you should be able to:

- The benefits of Azure policy
- The capabilities of Azure Automation
- How to use elastic jobs
- How to use Logic Apps

Introduction

Implementing HADR database platform solutions is more than just deploying a feature. Understand why and what you are doing to implement the right strategy.

Suppose you need to implement high availability and disaster recovery (HADR) for an IaaS virtual machine that is running SQL Server 2019. Do you know how much time you would have to bring things back online if there was a problem? Do you know what options are available in SQL Server and the reasons for choosing one option over another?

By the end of this module, you will have a solid foundation that will allow you to implement HADR solutions in Azure.

Learning objectives

After this module, you will understand:

- Recovery time objective (RTO)
- Recovery point objective (RPO)
- The available HADR options for both IaaS and PaaS
- How to devise a HADR strategy

Describe recovery time objective and recovery point objective

Understanding recovery time and recovery point objectives are crucial to your high availability and disaster recovery (HADR) plan as they are the foundation for any availability solution.

Recovery Time Objective

Recovery Time Objective (RTO) is the maximum amount of time available to bring resources online after an outage or problem. If that process takes longer than the RTO, there could be consequences such as financial penalties, work not able to be done, and so on. RTO can be specified for the whole solution, which includes all resources, as well as for individual components such as SQL Server instances and databases.

Recovery Point Objective

Recovery Point Objective (RPO) is the point in time to which a database should be recovered and equates to the maximum amount of data loss that the business is willing to accept. For example, suppose an IaaS VM containing SQL Server experiences an outage at 10:00 AM and the databases within the SQL Server instance have an RPO of 15 minutes. No matter what feature or technology is used to bring back that instance and its databases, the expectation is that there will be at most 15 minutes worth of data lost. That means the database can be restored to 9:45 AM or later to ensure minimal to no data loss meeting that stated RPO. There may be factors that determine if that RPO is achievable.

Defining Recovery Time and Recovery Point Objectives

RTOs and RPOs are driven by business requirements but are also based on various technological and other factors, such as the skills and abilities of the administrators (not just DBAs). While the business may want no downtime or zero data loss, that may not be realistic or possible for a variety of reasons. Determining your solution's RTO and RPO should be an open and honest discussion between all parties involved.

One of the aspects crucial for both RTO and RPO is knowing the cost of downtime. If you define that number and the overall effect being down or unavailable has to the business, it is easier to define solutions. For example, if the business can lose \$10,000 per hour or could be fined by a government agency if something could not be processed, that is a measurable way to help define RTO and RPO. Spending on the solution should be proportional to the amount, or the cost, of downtime. If your HADR solution costs \$X, but you wind up only being affected for a few seconds instead of hours or days when a problem occurs, it has paid for itself.

From a non-business standpoint, RTO should be defined at a component level (for example, SQL Server) as well as for the entire application architecture. The ability to recover from an outage is only as good as its weakest link. For example, if SQL Server and its databases can be brought online in five minutes but it takes application servers 20 minutes to do the same, the overall RTO would be 20 minutes, not five. The SQL Server environment could still have an RTO of five minutes; it still will not change the overall time to recover.

RPO deals specifically with data and directly influences the design of any HADR solution as well as administrative policies and procedures. The features used must support both the RTO and RPOs that are defined. For example, if transaction log backups are scheduled every 30 minutes but there is a 15-minute RPO, a database could only be recovered to the last transaction log backup available which in a best case would be 30 minutes ago. This timing assumes no other issues and the backups have been tested and are known to be good. While it is hard to test every backup generated for each database in your environment, backups are just files on a file system. Without

doing at least periodic restores, there is no guarantee they are good. Running checks during the backup process can give you some degree of confidence.

The specific features used, such as an Always On Availability Group (AG) or an Always On Failover Cluster Instance (FCI) will also affect your RTOs and RPOs. Depending on how the features are configured, IaaS or PaaS solutions may or may not automatically failover to another location, which could result in longer downtime. By defining RTO and RPO, the technical solution that supports that requirement can be designed knowing the allowances for time and data loss. If those wind-up being unrealistic, RTOs and RPOs must be adjusted accordingly. For example, if there is a desired RTO of two hours but a backup will take three hours to copy to the destination server for restoring, the RTO is already missed. These types of factors must be accounted for when determining your RTOs and RPOs.

There should be RTOs and RPOs defined for both HA and DR. HA is considered a more localized event that can be recovered from more easily. One example of high availability would be an AG automatically failing over from one replica to another within an Azure region. That may take seconds, and at that point, you would need to ensure that the application can connect after the failover. SQL Server's downtime would be minimal. A local RTO or RPO may potentially be measured in minutes depending on the critical nature of the solution or system.

DR would be akin to bringing up a whole new data center. There are lots of pieces to the puzzle; SQL Server is just one component. Getting everything online may take hours or longer. This is why the RTOs and RPOs are separate. Even if many of the technologies and features used for HA and DR are the same, the level of effort and time involved may not be.

All RTOs and RPOs should be formally documented and revised periodically or as needed. Once they are documented, you can then consider what technologies and features you may use for the architecture.

Explore high availability and disaster recovery options

To envision a solution for virtual machines (VMs), you must first understand the availability options for IaaS-based deployments.

Infrastructure-as-a-Service versus Platform-as-a-Service

When it comes to availability, the choice of IaaS or PaaS makes a difference. With IaaS, you have a virtual machine, which means there is an operating system with an installation of SQL Server. The administrator or group responsible for SQL Server would have a choice of high availability and disaster recovery (HADR) solutions and a great deal of control over what how that solution was configured.

With PaaS-based deployments such as Azure SQL Database, the HADR solutions are built into the feature and often just need to be enabled. There are minimal options that can be configured.

Because of these differences, the choice of IaaS or PaaS may influence the final design of your HADR solution.

SQL Server HADR Features for Azure Virtual Machine

When using IaaS, you can use the features provided by SQL Server to increase availability. In some cases, they can be combined with Azure-level features to increase availability even further.

The features available in SQL Server are shown in the table below

Feature Name	Protects
Always On Failover Cluster Instance (FCI)	Instance
Always On Availability Group (AG)	Database
Log Shipping	Database

An instance of SQL Server is the entire installation of SQL Server (binaries, all the objects inside the instance including things like logins, SQL Server Agent jobs, and databases). Instance-level protection means that the entire instance is accounted for in the availability feature.

A database in SQL Server contains the data that end users and applications use. There are system databases that SQL Server relies on, as well as databases created for use by end users and applications. An instance of SQL Server always has its own system databases. Database-level protection means that anything that is in the database, or is captured in the transaction log for a user or application database, is accounted for as part of the availability feature. Anything that exists outside of the database or that is not captured as part of the transaction log such as SQL Server Agent jobs and linked servers must be manually dealt with to ensure the destination server can function like the primary if there is a planned or unplanned failover event.

Both FCIs and AGs require an underlying cluster mechanism. For SQL Server deployments running on Windows Server, it is a Windows Server Failover Cluster (WSFC) and for Linux it is Pacemaker.

Always On Failover Cluster Instances

An FCI is configured when SQL Server is installed. A standalone instance of SQL Server cannot be converted to an FCI. The FCI is assigned a unique name as well as an IP address that is different from the underlying servers, or nodes, participating in the cluster. The name and IP address must also be different from the underlying cluster mechanism. Applications and end users would use the unique name of the FCI for access. This abstraction enables applications to not have to know where the instance is running. One major difference between Azure-based FCIs versus on-premises FCIs, is that for Azure, an internal load balancer (ILB) is required. The ILB is used to help ensure applications and end users can connect to the FCI's unique name.

When an FCI fails over to another node of a cluster, whether it is initiated manually or happens due to a problem, the entire instance restarts on another node. That means the failover process is a full stop and start of SQL Server. Any applications or end users connected to the FCI will be disconnected during failover and only applications that can handle and recover from this interruption can reconnect automatically.

Upon starting up on the other node, the instance goes through the recovery process. The FCI will be consistent to the point of failure, so technically there will be no data loss but any transactions that need to be rolled back will do so as part of recovery. As noted above, because this is instance-level protection, everything necessary (logins, SQL Server Agent jobs, etc.) is already there so business can continue as usual once the databases are ready.

FCIs require one copy of a database, but that is also its single point of failure. To ensure another node can access the database, FCIs require some form of shared storage. For Windows Server-based architectures, this can be achieved via an Azure Premium File Share, iSCSI, Azure Shared Disk, Storage Spaces Direct (S2D), or a supported third-party solution like SIOS DataKeeper. FCIs using Standard Edition of SQL Server can have up to two nodes. FCIs also require the use of Active Directory Domain Services (AD DS) and Domain Name Services (DNS), so that means AD DS and DNS must be implemented somewhere in Azure for an FCI to work.

Using Windows Server 2016 or later, FCIs can use Storage Replica to create a native disaster recovery solution for FCIs without having to use another feature such as log shipping or AGs.

Always On availability groups

AGs were introduced in SQL Server 2012 Enterprise Edition and as of SQL Server 2016, are also in Standard Edition. In Standard Edition, an AG can contain one database whereas in Enterprise Edition, an AG can have more than one database. While AGs share some similarities with FCIs, in most ways they are different.

The biggest difference between an FCI and an AG is that AGs provide database-level protection. The primary replica is the instance participating in an AG that contains the read/write databases. A secondary replica is where the primary sends transactions over the log transport to keep it synchronized. Data movement between a primary replica can be synchronous or asynchronous. The databases on any secondary replica are in a loading state, which means they can receive transactions but cannot be a fully writeable copy until that replica becomes the primary. An AG in Standard Edition can have at most two replicas (one primary, one secondary) whereas Enterprise Edition supports up to nine (one primary, eight secondary). A secondary replica is initialized either from a backup of the database, or as of SQL Server 2016, you can use a feature called 'automatic seeding'. Automatic seeding uses the log stream transport to stream the backup to the secondary replica for each database of the availability group using the configured endpoints.

An AG provides abstraction with the listener. The listener functions like the unique name assigned to an FCI and has its own name and IP address that is different from anything else (WSFC, node, etc.). The listener also requires an ILB and goes through a stop and start. Applications and end users can use the listener to connect, but unlike an FCI, if desired, the listener does not have to be used. Connections directly to the instance can occur. With Enterprise Edition, secondary replicas in Enterprise Edition can also be configured for read-only access if desired and can be used for other functionality such as database consistency checks (DBCCs) and backups.

AGs can have a quicker failover time compared to an FCI, which is one reason they are attractive. While AGs do not require shared storage, each replica has a copy of the data, which increases the total number of copies of the database and overall storage costs. The storage is local to each replica. For example, if the data footprint of the databases on the primary replica is 1 TB, each replica will also have the same. If there are five replicas, that means you need 5 TB of space.

Remember that any object that exists outside of the database or is not captured in the database's transaction log must manually be created and accounted for on any other SQL Server instance should that instance need to become the new primary replica. Examples of objects you would be responsible for include SQL Server Agent jobs, instance-level logins, and linked servers. If you can use Windows authentication or use contained databases with AGs, it will simplify access.

Many organizations may face challenges implementing highly available architectures, and may only need the high availability provided by the Azure platform, or using a PaaS solution like Azure SQL Managed Instance. Before we look at Azure platform solutions, there is one other SQL Server feature that you should know about: log shipping.

Log shipping

Log shipping has been around since the early days of SQL Server. The feature is based on backup, copy, and restore and is one of the simplest methods of achieving HADR for SQL Server. Log shipping is primarily used for disaster recovery, but it could also be used to enhance local availability.

Log shipping, like AGs, provides database-level protection, which means you still need to account for SQL Server Agent jobs, linked servers, instance-level logins, etc. There is no abstraction provided natively by log shipping, so a switch to another server participating in log shipping must be able to tolerate a name change. If that is not possible, there are methods such as a DNS alias, which can be configured at the network layer to try to mitigate the name change issues.

The log shipping mechanism is simple: first, take a full backup of the source database on the primary server, restore it in a loading state (STANDBY or NORECOVERY) on another instance known as a secondary server or warm standby. This new copy of the database is known as a secondary database. An automated process built into SQL Server will then automatically backup the primary database's transaction log, copy the backup to the standby server, and finally, restore the backup onto the standby.

The SQL Server HADR features are not the only options to enhance IaaS availability. There are some features in Azure that should also be considered.

Describe Azure high availability and disaster recovery features for Azure Virtual Machines

Azure provides three main options to enhance availability for IaaS deployments:

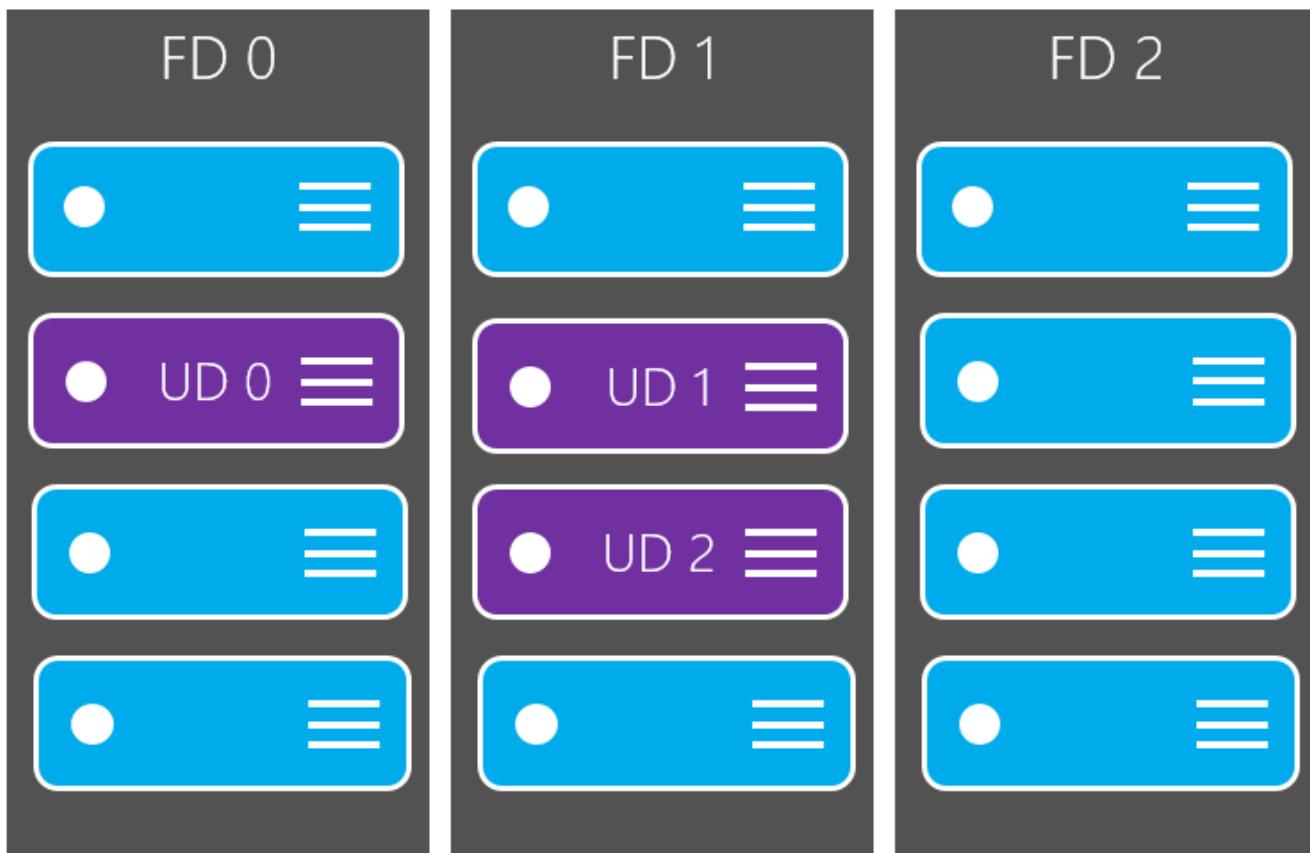
- Availability Sets
- Availability Zones
- Azure Site Recovery

All three of these options are external to the virtual machine (VM) and do not know what kind of workload is running inside of it.

Availability sets

Availability sets provide uptime against Azure-related maintenance and single points of failure in a single data center. This was one of the first availability features introduced into the Azure platform, and effectively it can be thought of as anti-affinity rules for your VMs. This means if you had two SQL Server VMs in an availability set or log shipping pair, they would be guaranteed to never run on the same physical server.

Availability sets are separated into both fault domains and update domains to support both updates to the underlying Azure Infrastructure. Fault domains are sets of servers within a data center, which use the same power source and network. There can be up to three fault domains in a data center as depicted in the image below by FD 0, 1, and 2. Update domains, denoted by UD in the image below, indicate groups of virtual machines and underlying physical hardware that can be rebooted at the same time. Different update domains ensure separation.



Availability sets and zones do not protect against in-guest failures, such as an OS or RDBMS crash; which is why you need to implement additional solutions such as AGs or FCIs to ensure you meet RTOs and RPOs. Both availability sets and zones are designed to limit the impact of environmental problems at the Azure level such as datacenter failure, physical hardware failure, network outages, and power interruptions.

For a multi-tier application, you should put each tier of the application into its own availability set. For example, if you were building a web application that has a SQL Server backend along with Active Directory Domain Services (AD DS), you would create an availability set for each tier (web, database, and AD DS).

Availability sets is not the only way to separate IaaS VMs. Azure also provides Availability Zones, but the two cannot be combined. You can pick one or the other.

Availability zones

Availability zones account for data center-level failure in Azure. Each Azure region consists of many data centers with low latency network connections between them. When you deploy VM resources in a region that supports Availability Zones, you have the option to deploy those resources into Zone 1,2, or 3. A zone is a unique physical location, that is, a data center, within an Azure region.

Zone numbers are logical representations. For example, if two Azure subscribers both deploy a VM into Zone 1 in their own subscriptions, that does not mean those VMs exist in the same physical Azure data center. Additionally, because of the distance there can be some additional latency introduced into zonal deployments. You should test the latency between your VMs to ensure that the latency meets performance targets. In most cases round-trip latency will be less than 1 millisecond, which supports synchronous data movement in features like availability groups. You can also deploy Azure SQL Database into Availability Zones.

Azure Site Recovery

Azure Site Recovery provides enhanced availability for VMs at the Azure level and can work with VMs hosting SQL Server. Azure Site Recovery replicates a VM from one Azure region to another to create a disaster recovery solution for that VM. As noted earlier, this feature does not know that SQL Server is running in the VM and knows nothing about transactions. While Azure Site Recovery may meet RTO, it may not meet RPO since it is not accounting for where data is inside SQL Server. Azure Site Recovery has a stated monthly RTO of two hours. While most database professionals may prefer to use a database-based method for disaster recovery, Azure Site Recovery works well if it meets your RTO and RPO needs.

Describe high availability and disaster recovery options for PaaS deployments

PaaS is different when it comes to availability; you can only configure the options that Azure provides.

For the SQL Server-based options of Azure SQL Database and Azure SQL Database Managed Instance, the options are active geo-replication (Azure SQL Database only) and autofailover groups (Azure SQL Database or Azure SQL Database Managed Instance).

Azure Database for MySQL has a service level agreement, which guarantees availability of 99.99, meaning nearly no downtime should be encountered. For Azure Database for MySQL, if a node-level problem happens such as hardware failure, a built-in failover mechanism will kick in. All transactional changes to the MySQL database are written synchronously to storage upon commit. If a node-level interruption occurs, the database server automatically creates a new node and attaches the data storage.

From an application standpoint, you will need to code the necessary retry logic because all connections are dropped as part of spinning up the new node and any in flight transactions are lost. This process is considered a best practice for any cloud application, as they should be designed to handle transient failures.

Azure Database for PostgreSQL uses a similar model to MySQL in its standard deployment model; however, Azure PostgreSQL also offers a scale-out hyperscale solution called Citus. Citus provides both scale-out and additional high availability for a server group. If enabled, a standby replica is configured for every node of a server group, which would also increase cost since it would double the number of servers in the group. In the event, the original node has a problem such as becoming unresponsive or failing completely, the standby takes its place. The data is kept in sync via PostgreSQL synchronous streaming replication.

As with Azure Database for MySQL, solutions that use Azure Database for PostgreSQL must also include retry logic in the application because of dropped connections and loss of in-flight transactions.

Both Azure Database for MySQL and PostgreSQL supports the option for a read replica. This means a replica can be used for activities like reporting to offload work from the primary database. A read replica also enhances availability because it exists in another region.

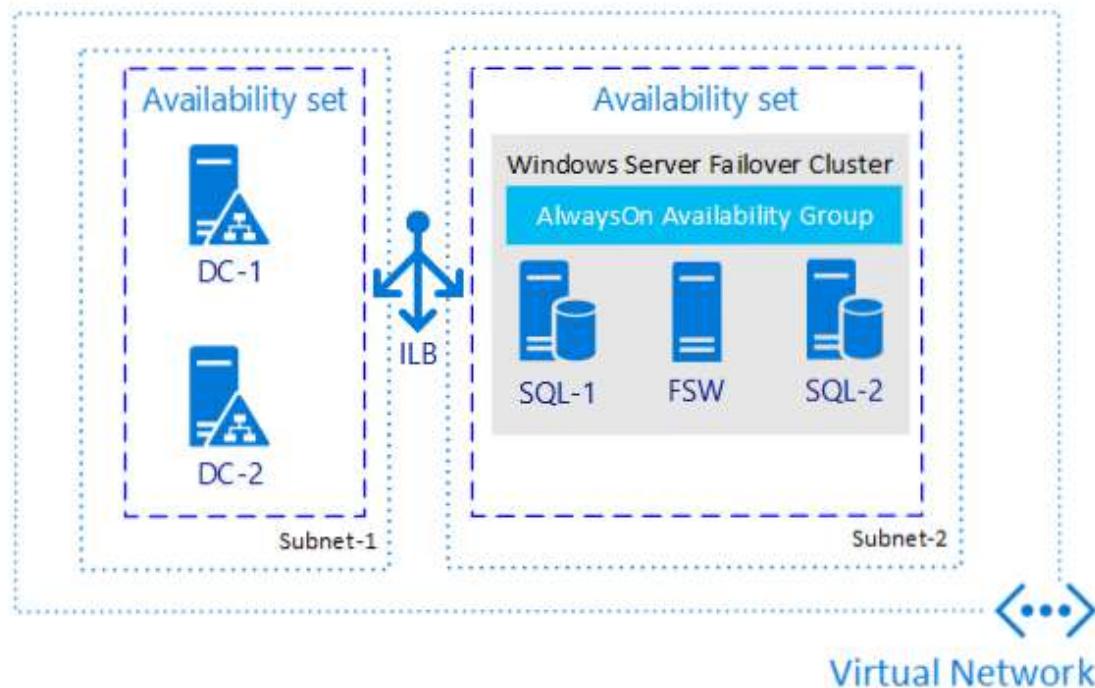
This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Explore an IaaS high availability and disaster recovery solution

There are many different combinations of features that could be deployed in Azure for IaaS. This section will cover five common examples of SQL Server high availability and disaster recovery (HADR) architectures in Azure.

Single Region High Availability Example 1 – Always On availability groups

If you only need high availability and not disaster recovery, configuring an (availability group) AG is one of the most ubiquitous methods no matter where you are using SQL Server. The image below is an example of what one possible AG in a single region could look like.



Why is this architecture worth considering?

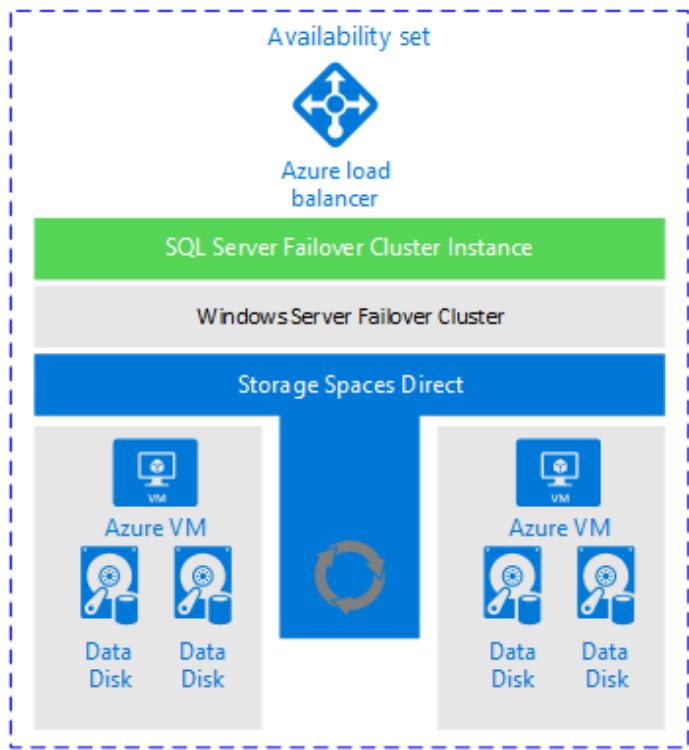
- This architecture protects data by having more than one copy on different virtual machines (VMs).
- This architecture allows you to meet recovery time objective (RTO) and recovery point objective (RPO) with minimal-to-no data loss if implemented properly.
- This architecture provides an easy, standardized method for applications to access both primary and secondary replicas (if things like read-only replicas will be used).
- This architecture provides enhanced availability during patching scenarios.
- This architecture needs no shared storage, so there is less complication than when using a failover cluster instance (FCI).

Single Region High Availability Example 2 – Always On Failover Cluster Instance

Until AGs were introduced, FCIs were the most popular way to implement SQL Server high availability. FCIs, however, were designed when physical deployments were dominant. In a virtualized world, FCIs do not provide many of the same protections in the way they would on physical hardware because it is rare for a VM to have a

problem. FCIs were designed to protect against things like network card failure or disk failure, both of which would likely not happen in Azure.

Having said that, FCIs do have a place in Azure. They work, and as long as you have the right expectations about what is and is not provided, an FCI is a perfectly acceptable solution. The image below, from the Microsoft documentation, shows a high-level view of what an FCI deployment looks like when using Storage Spaces Direct.

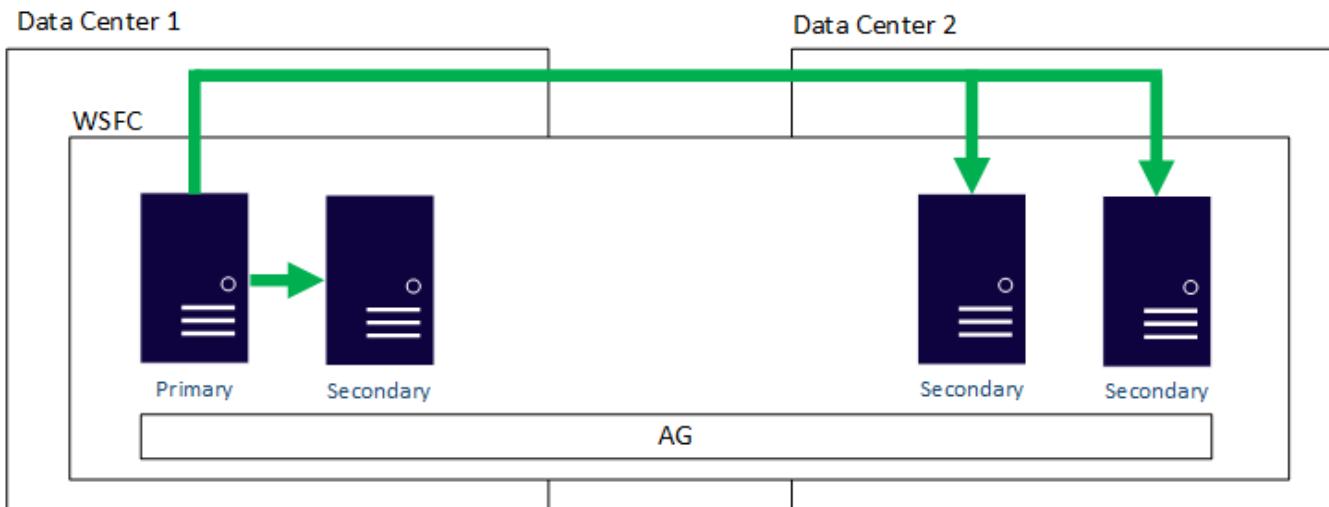


Why is this architecture worth considering?

- FCIs are still a popular availability solution.
- The shared storage story is improving with feature like Azure Shared Disk.
- This architecture meets most RTO and RPO for HA (although DR is not handled).
- This architecture provides an easy, standardized method for applications to access the clustered instance of SQL Server.
- This architecture provides enhanced availability during patching scenarios.

Disaster Recovery Example 1 – Multi-Region or Hybrid Always On availability group

If you are using AGs, one option is to configure the AG across multiple Azure regions or potentially as a hybrid architecture. This means that all nodes which contain the replicas participate in the same WSFC. This assumes good network connectivity, especially if this is a hybrid configuration. One of the biggest considerations would be the witness resource for the WSFC. This architecture would require AD DS and DNS to be available in every region and potentially on premises as well if this is a hybrid solution. The image below shows what a single AG configured over two locations looks like using Windows Server.



Why is this architecture worth considering?

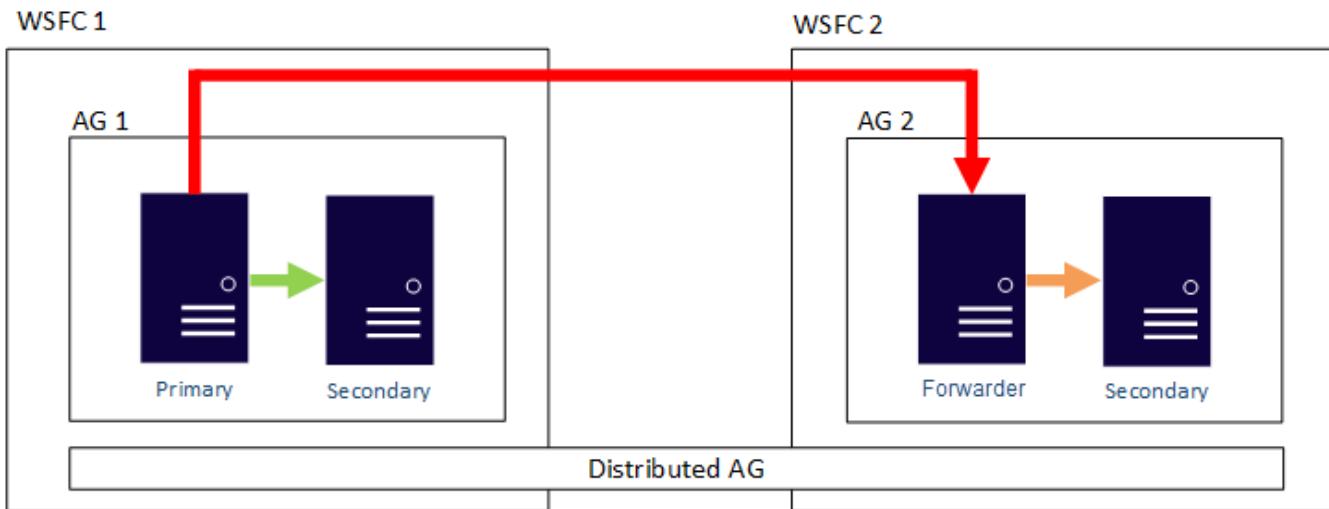
- This architecture is a proven solution; it is no different than having two data centers today in an AG topology.
- This architecture works with Standard and Enterprise editions of SQL Server.
- AGs naturally provide redundancy with additional copies of data.
- This architecture makes use of one feature that provides both HA and D/R

Disaster Recovery Example 2 –Distributed availability group

A distributed AG is an Enterprise Edition only feature introduced in SQL Server 2016. It is different than a traditional AG. Instead of having one underlying WSFC where all of nodes contain replicas participating in one AG as described in the previous example, a distributed AG is made up of multiple AGs. The primary replica containing the read write database is known as the global primary. The primary of the second AG is known as a forwarder and keeps the secondary replica(s) of that AG in sync. In essence, this is an AG of AGs.

This architecture makes it easier to deal with things like quorum since each cluster would maintain its own quorum, meaning it also has its own witness. A distributed AG would work whether you are using Azure for all resources, or if you are using a hybrid architecture.

The image below shows an example distributed AG configuration. There are two WSFCs. Imagine each is in a different Azure region or one is on premises and the other is in Azure. Each WSFC has an AG with two replicas. The global primary in AG 1 is keeping the secondary of replica of AG 1 synchronized as well as the forwarder, which also is the primary of AG 2. That replica keeps the secondary replica of AG 2 synchronized.

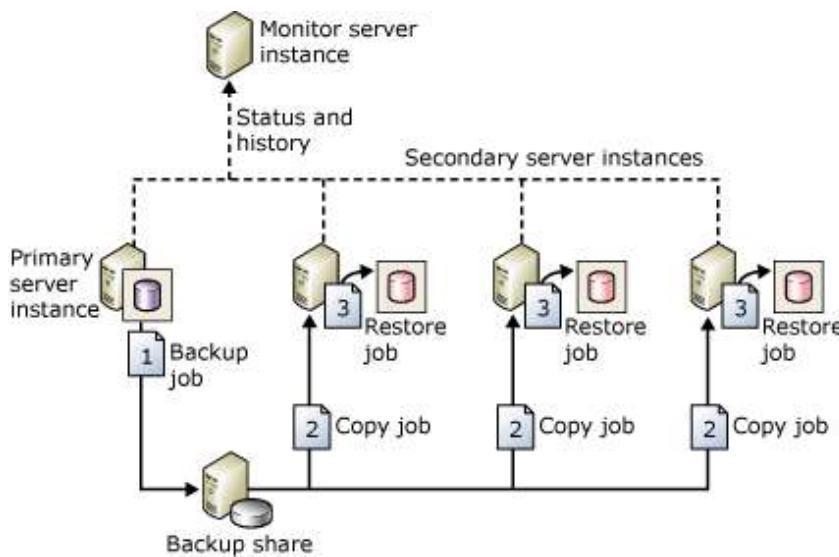


Why is this architecture worth considering?

- This architecture separates out the WSFC as a single point of failure if all nodes lose communication
- In this architecture, one primary is not synchronizing all secondary replicas.
- This architecture can provide failing back from one location to another.

Disaster Recovery Example 3 – Log shipping

Log shipping is one of the oldest HADR methods for configuring disaster recovery for SQL Server. As described above, the unit of measurement is the transaction log backup. Unless the switch to a warm standby is planned to ensure no data loss, data loss will most likely occur. When it comes to disaster recovery, it is always best to assume some data loss even if minimal. The image below, from the Microsoft documentation, shows an example log shipping topology.



Why is this architecture worth considering?

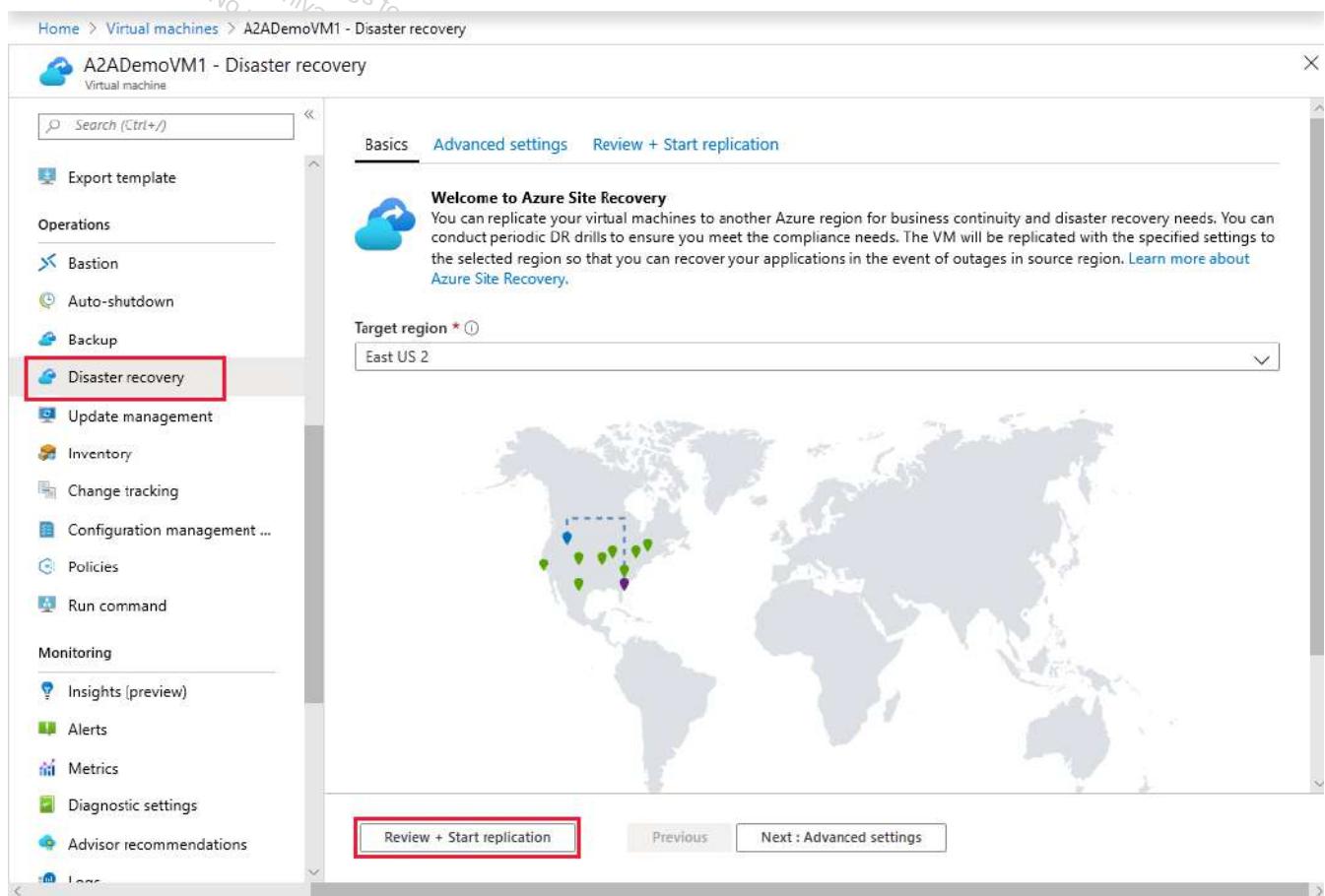
- Log shipping is a tried-and-true feature that has been around for over 20 years
- Log shipping is easy to deploy and administer since it is based on backup and restore.
- Log shipping is tolerant of networks that are not robust.

- Log shipping meets most RTO and RPO goals for DR.
- Log shipping is a good way to protect FCIs.

Disaster Recovery Example 4 – Azure Site Recovery

For those who do not want to implement a SQL Server-based disaster solution, Azure Site Recovery is a potential option. However, most data professionals prefer a database-centric approach as it will generally have a lower RPO.

The image below, from the Microsoft documentation, shows where in the Azure portal you would configure replication for Azure Site Recovery.



Why is this architecture worth considering?

- Azure Site Recovery will work with more than just SQL Server.
- Azure Site Recovery may meet RTO and possibly RPO.
- Azure Site Recovery is provided as part of the Azure platform.

Describe hybrid solutions

You should now understand recovery time objectives (RTOs) and recovery point objectives (RPOs) as well as the different features both in SQL Server as well as Azure to increase availability, you can put all of that together and design a solution to meet your high availability and disaster recovery (HADR) requirements.

While an architecture can be deployed in one or more Azure regions, many organizations will need or want to have solutions that span both on premises and Azure, or possibly Azure to another public cloud. This type of architecture is known as a hybrid solution.

PaaS solutions by nature are not designed to allow traditional hybrid solutions. HADR is provided by the Azure infrastructure. There are some exceptions. For example, SQL Server's transactional replication feature can be configured from a publisher located on premises (or another cloud) to an Azure SQL Database Managed Instance subscriber, but not the other way.

Hybrid solutions are therefore IaaS-based since they rely on traditional infrastructure. Hybrid solutions are useful. Not only can they be used to help migrate to Azure, but the most common usage of a hybrid architecture is to create a robust disaster recovery solution for an on-premises system. For example, a secondary replica for an AG can be added in Azure. That means any associated infrastructure must exist, such as AD DS and DNS.

Arguably the most important consideration for a hybrid HADR solution that extends to Azure is networking. Not having the right bandwidth could mean missing your RTO and RPO. Azure has a fast networking option called ExpressRoute. If ExpressRoute is not something your company can or will implement, configure a secure site-to-site VPN so that the Azure VMs will act as an extension of your on-premises infrastructure. Exposing IaaS VMs directly to the public internet is discouraged.

Although not traditionally thought of as hybrid, Azure can also be used as the destination for a database backup and as cold, archival storage for backups.

Knowledge check

Multiple choice

What is RPO?

- The number of nodes in a cluster
- The point to which data needs to be recovered after a failure
- A partial database restore

Check Answers

Multiple choice

What is a hybrid solution?

- A solution that has resources both in Azure as well as on premises or in another cloud provider
- A solution that uses two different database engines, for example, MySQL and SQL Server
- A solution that spans two different versions of SQL Server

Check Answers

Multiple choice

What is available after failover with database-level protection in SQL Server?

- Logins, Databases, and SQL Server Agent Job
- Databases and SQL Server Agent jobs
- Whatever is in the databases; anything outside must be dealt with manually

Check Answers

Summary

Deploying the right HADR solution in Azure is more than just picking a feature. A solution must meet the requirements, and specifically, the RTO and RPO expected by the business. Depending on the path (IaaS or PaaS), there could be multiple options to choose from.

Now that you've reviewed this module, you should be able to:

- Explain recovery time objective (RTO)
- Explain recovery point objective (RPO)
- Explain the available HADR options for both IaaS and PaaS
- Devise a HADR strategy

Introduction

If you're using IaaS for deploying your database solution, there are considerations for properly deploying HADR.

Suppose you want to use an Availability Group (AG). How is deploying a Windows Server Failover Cluster (WSFC) different? Are there any specific considerations for AGs that differ from an on-premises solution?

Implementing a PaaS-based high availability and disaster recovery (HADR) solution for PaaS solutions is different than IaaS. For IaaS, the configuration is done at the Azure level. For PaaS, the solutions that allow your applications and data to be highly available and meet your RTOs and RPOs are configured within the database or database server.

By the end of this module, you'll understand what is needed to be able to deploy IaaS database platform solutions in Azure.

Learning objectives

In this module, you'll learn:

- What to consider when deploying a WSFC in Azure
- What to consider when deploying an AG in Azure
- How to enable AGs
- Temporal Tables
- Active-geo replication
- Auto-failover groups

Describe failover clusters in Windows Server

For all availability configurations of availability groups (AGs), an underlying cluster is required. There are many aspects of setting up a cluster that you need to be aware of.

Considerations for a Windows Server Failover Cluster in Azure

Deploying a Windows Server Failover Cluster (WSFC) in Azure is similar to configuring one on premises. There are some Azure-specific considerations, which is what this section will focus on.

One of the most important aspects is deciding what to use for a witness resource. Witness is a core component of the quorum mechanism. Quorum is what helps ensure that everything in the WSFC stays up and running. If you lose quorum, the WSFC will go down taking an Availability Group (AG) or Failover Cluster Instance (FCI) with it. The witness resource can be a disk, file share (SMB 2.0 or later), or cloud witness. It's recommended that you use a cloud witness as it is fully Azure-based, especially for solutions that will span multiple Azure regions or are hybrid. The cloud witness feature is available in Windows Server 2016 and later.

The next consideration is the Microsoft Distributed Transaction Coordinator (DTC or MSDTC). Some applications use it, but most applications don't. If you require DTC and are deploying an AG or FCI, you should cluster DTC. Clustering DTC requires a shared disk to work properly even though you may not require one otherwise, as in the case of an AG.

Most WSFC deployments require the use of both AD DS and DNS; FCIs always do. AGs can be configured without requiring AD DS but still needing DNS. In Windows Server 2016, there's a variant of a WSFC called a Workgroup Cluster, which can be used with AGs only.

The WSFC itself needs a unique name in the domain (and DNS) and requires an object in AD DS called the cluster name object (CNO). Anything created in context of the WSFC that has a name will require a unique name, and at least one IP address. If the configuration will stay in a single region, IP addresses will be in a single subnet. If the WSFC will span multiple regions, more than one IP address will be associated with the WSFC, and an AG if it spans multiple regions as part of the WSFC.

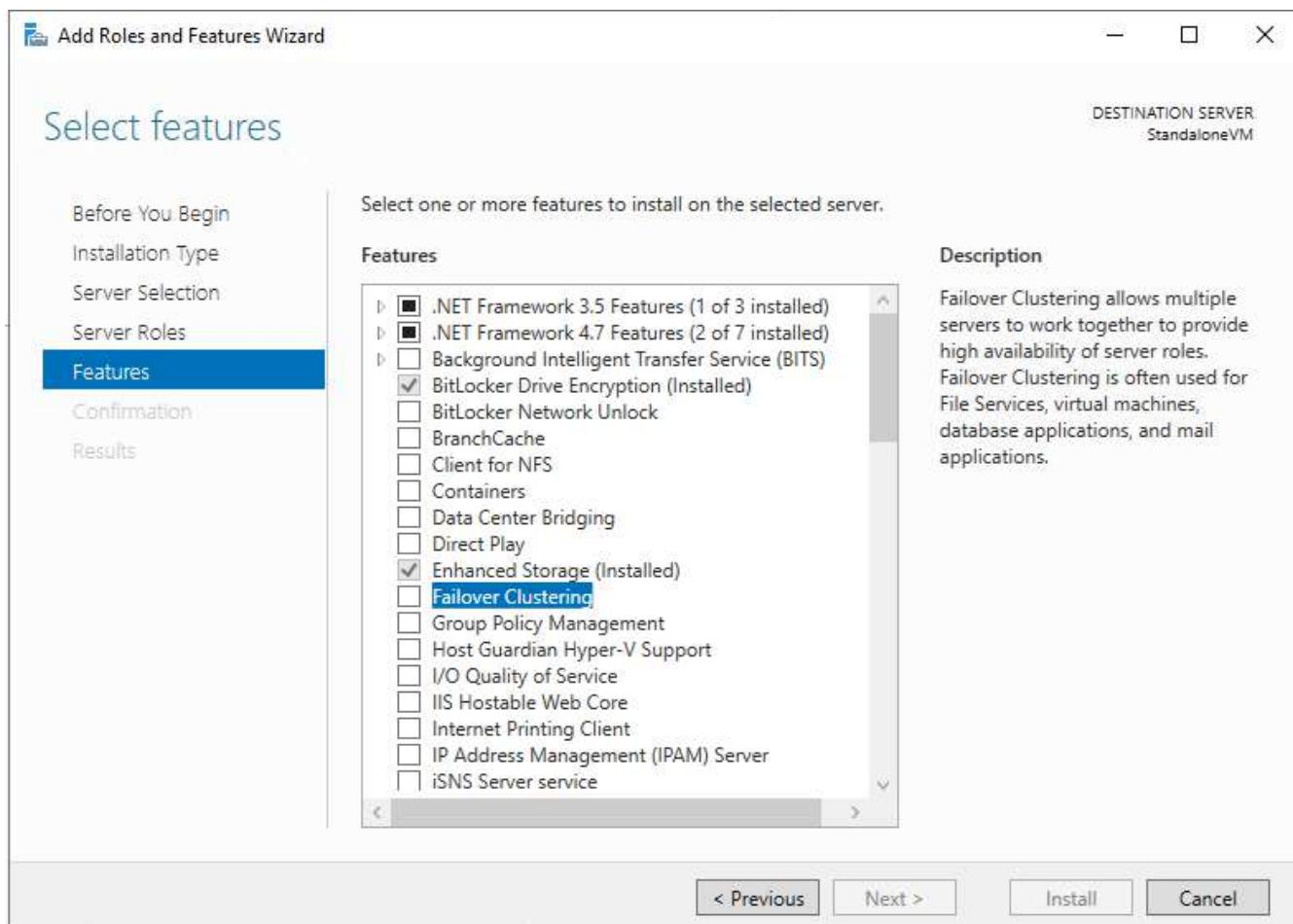
A typical Azure-based WSFC will only require a single virtual network card (vNIC). Unlike the setup for an on-premises physical or virtual server, the IP address for the vNIC has to be configured in Azure, not in the VM. That means inside the VM it will show up as a dynamic (DHCP) address. This is expected, however cluster validation will generate a warning that can safely be ignored.

Considerations for the WSFC's IP address is also different from on premises. There's no way to reserve the IP address at the Azure level since it's fully maintained within the guest configuration. This means that if subsequent resources that use an IP address in Azure use DHCP, you must check for conflicts.

The failover clustering feature

Before a WSFC can be configured, the underlying Windows feature must be enabled on every node that will participate in the WSFC. This can be done in one of two ways: using the Server Manager or via PowerShell.

In Server Manager, Failover Clustering must be enabled in the Add Roles and Features Wizard.



To enable the feature via PowerShell, use the following command, which will also install the utilities, which will be used to administer a WSFC:

```
Install-WindowsFeature Failover-Clustering -IncludeManagementTools
```

Once the feature is installed on the servers targeted as WSFC nodes, you must then validate the configuration.

Cluster validation

For a WSFC to be considered supported, it must pass cluster validation. Cluster validation is a built-in process that checks the nodes via a series of tests to ensure that the entire configuration is suitable for clustering. After the validation is complete, each of the tests will come back with an error, a warning, a pass, or a message that the test isn't applicable. Warnings are acceptable if that condition is expected in your environment. If not, it should be addressed. All errors must be resolved.

Validation can be run via Failover Cluster Manager or by using the Test-Cluster PowerShell cmdlet.

For FCIs, these tests also check the shared storage to ensure that the disks are configured correctly. For AGs with no shared storage, in Windows Server 2016 and later, the results will come back as not applicable. For Windows Server 2012 R2, the disk tests will show a warning when there are no shared disks. This state is expected.

Once the configuration is deemed valid by the cluster validation process, you can then create the WSFC.

Create a Windows Server Failover Cluster

To create a WSFC properly in Azure, you can't use the Wizard in Failover Cluster Manager for FCIs or AGs deployed using Windows Server 2016 and earlier. Due to the DHCP issue mentioned earlier, currently the only way to create the WSFC is to use PowerShell and specify the IP address. This could change in future versions of Windows Server.

For a configuration that has shared storage, use the following syntax:

```
New-Cluster -Name MyWSFC -Node Node1,Node2,...,NodeN -StaticAddress w.x.y.z
```

where MyWSFC is the name of the WSFC you want, Node1, Node2,..., NodeN are the names of the nodes that will participate in the WSFC, and w.x.y.z is the IP address of the WSFC (for example: 10.252.1.100.) If you're creating a WSFC that spans multiple subnets, you can specify more than one IP address for -StaticAddress separated by commas.

For a configuration that doesn't have shared storage, add the -NoStorage option.

```
New-Cluster -Name MyWSFC -Node Node1,Node2,...,NodeN -StaticAddress w.x.y.z -NoStorage
```

For a Workgroup Cluster that will only use DNS, the syntax is also slightly different.

```
New-Cluster -Name MyWSFC -Node Node1,Node2,...,NodeN -StaticAddress w.x.y.z -NoStorage -AdministrativeAccessPoint DNS
```

Windows Server 2019 by default will use a distributed network name for IaaS. A distributed network name is one that creates just a network name, but the IP address is tied to the underlying nodes. You no longer need to specify an IP address as shown above if it isn't needed or necessary. A distributed network name is for the WSFC's name only; it can't be used with the name of an AG or FCI.

The WSFC creation mechanism in Windows Server 2019 detects if it's running in Azure or not and will create the cluster using a distributed network name unless you specify it differently. However, there are cases you may want to consider deploying a WSFC traditionally using PowerShell. For this, you need to add one more option: `-ManagementPointNetwork` with a value of `singleton`. An example would look like this:

```
New-Cluster -Name MyWSFC -Node Node1,Node2,...,NodeN -StaticAddress w.x.y.z -NoStorage -ManagementPointNetwork singleton
```

For a Workgroup Cluster, you'll need to ensure the name and IP address(es) are in DNS for any name or IP address created in the context of the WSFC such as the WSFC itself, an FCI name and IP address, and an AG listener name and IP address.

With Windows Server 2019, Microsoft changed how WSFCs are created by default in Azure. Instead of creating a network name and an IP address, it uses a distributed network name.

Failover Cluster Instance

Failover Cluster Instances uses Windows Server Failover Clustering (WSFC) functionality to provide high availability through redundancy at the instance level.

An FCI is an instance of SQL Server that is installed across WSFC nodes and, possibly, across multiple subnets. On the network, an FCI appears as a single local instance that uses a virtual network name. Therefore, it becomes

transparent for the application, since it doesn't know which node the instance is currently running on. As a result, there is no need to reconfigure clients and applications during or after a failover.

The multi-subnet support eliminates the need for a virtual LAN, which increases the protection and flexibility of a multi-subnet FCI.

When using multi-subnet, each subnet of the FCI is assigned a virtual IP address, and a failover occurs as follows.

- Virtual network names on DNS server are updated with virtual IP addresses corresponding to the respective subnets
- After a multi-subnet failover, clients and applications can then connect to the FCI via the same virtual network name.

To learn more about how multi-subnet works on FCI, see [SQL Server Multi-Subnet Clustering \(SQL Server\)](#).

Distributed Network Name (DNN)

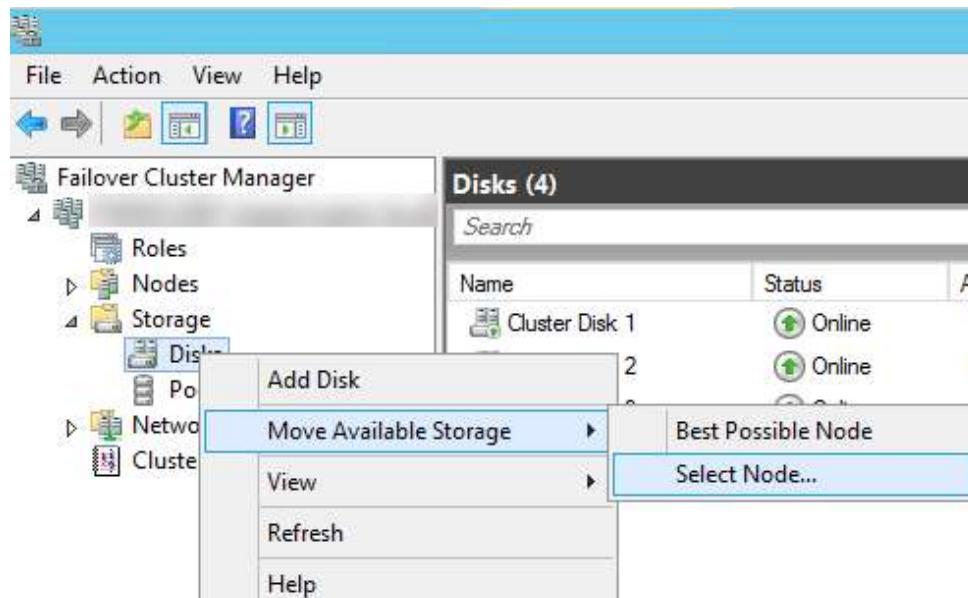
The distributed network name (DNN) replaces the virtual network name (VNN) as the connection point when used with FCI. As a result, the VNN no longer requires an Azure Load Balancing service.

The VNN still exists in an FCI deployment, but the clients connect to the DNN DNS name instead of the VNN name.

To learn how to configure a distributed network name for FCI, see [Configure a DNN for failover cluster instance](#).

Test the Windows Server Failover Cluster

After the WSFC is created but before configuring FCIs or AGs, test that the WSFC is working properly. For clusters that require shared storage, such as for those supporting a SQL Server Failover Cluster Instance, it's important that you test the ability for all of the nodes in the cluster to access any shared storage, and to take ownership of the storage as they would in a failover. You can do this by clicking on Storage, then Disks in Failover Cluster Manager, as shown in the image below. If you right-click on each clustered disk device, you'll see the option **Move Available Storage**. If you choose the Select Node option, you can force the storage to fail over to the other nodes in the cluster to confirm this functionality.



Now that you understand the considerations for a WSFC in Azure, let's look at the considerations for deploying the Always On features on top of a WSFC in Azure.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

Configure Always-on availability groups

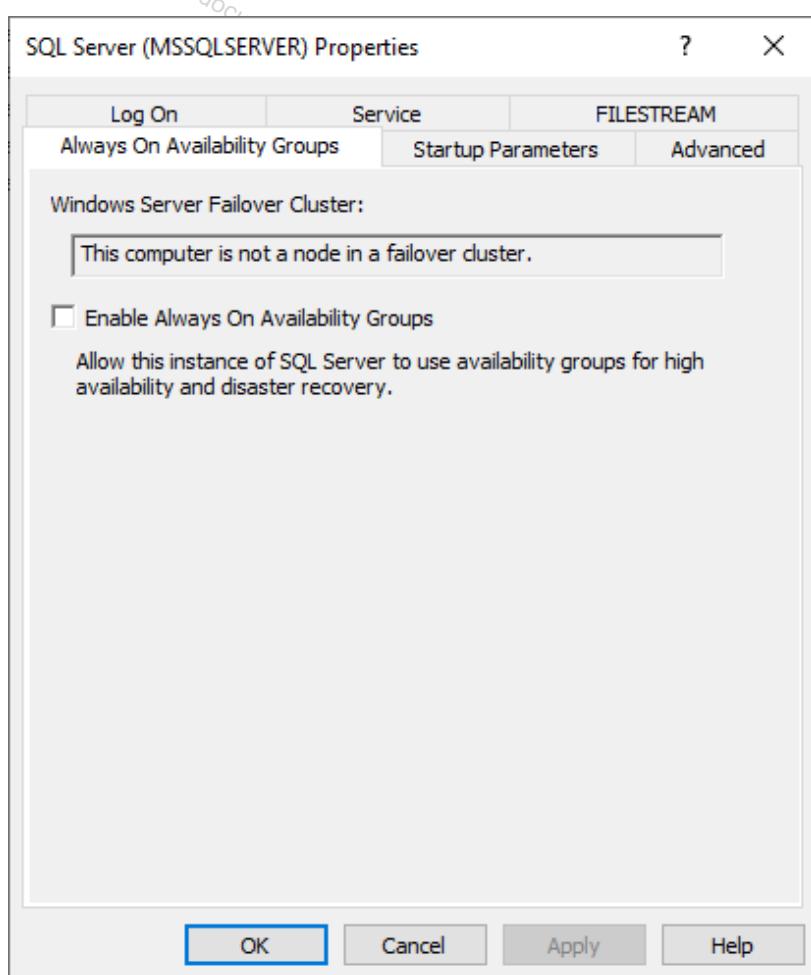
For all availability configurations of availability groups (AGs), an underlying cluster is required, whether or not it uses AD DS. By the end of this unit, you'll understand the considerations for deploying an AG in Azure.

Considerations for Always On availability groups in Azure

Configuring an AG is nearly the same in Azure as it is on premises as are most of the considerations, such as how to initialize secondary replicas. Most of the Azure-specific considerations were discussed earlier, such as needing an ILB. Same as the WSFC itself, you can't reserve the listener's IP address in Azure so you need to ensure something else doesn't come along and grab it otherwise there could be a conflict on the network, which in turn could cause availability headaches.

Don't place any permanent database on the ephemeral storage. All virtual machines (VMs) that are participating in an AG should have the same storage configuration. You must size disks appropriately for performance depending on the application workload.

Before an AG can be configured, the AG feature must be enabled. This can be done in SQL Server Configuration Manager as shown in the image below or via PowerShell with the cmdlet [Enable-SqlAlwaysOn](#). Enabling the AG feature will require a stop and start of the SQL Server service.



Create the Availability group

Creating an AG in Azure is the same as it is on premises. SQL Server Management Studio (SSMS), T-SQL, or PowerShell can be used.

The only difference is that whether or not you create the listener as part of the initial AG configuration, as the listener requires the creation of an Azure load balancer and has some extra configuration in the WSFC related to the load balancer.

Create an Internal Azure load balancer

Once the listener is created, an internal load balancer (ILB) must be used. Without configuring an ILB, applications, end users, administrators, and others can't use the listener unless they were connected to the VM that hosts an AG's primary replica.

You can use a basic or a standard load balancer depending on your preference or configuration. Deployments using Availability Zones require the use of a standard load balancer. The listener IP address and the port used for the listener are what is configured as part of the load balancer. A single load balancer supports more than one IP address, so depending on your standards, you may not need a different load balancer for each AG configured on those nodes.

Another consideration for the load balancer is the probe port. Without the probe port, the listener won't work properly as it isn't enough just to create the load balancer. Each IP address that will use the load balancer requires a unique probe port. If there are going to be two listeners, there must be two probe ports. Probe ports are high numbers such as 59999.

The probe port is set on the IP address(es) associated with the listener with the following syntax:

```
Get-ClusterResource IPAddressResourceNameForListener | Set-ClusterParameter ProbePort  
PortNumber
```

Adding the probe port will require a stop and start of the IP address of the listener, which will also temporarily cause the AG to be brought offline, so it's best to get this configured before deploying in production.

If you have a multi-subnet configuration, a load balancer will need to be configured in each subnet (whether or not the other subnet is deployed to different region) and the probe port for that region associated with the IP resource for that subnet in the WSFC.

If you can't directly connect to the listener, then you need to use the PowerShell cmdlet Test-NetConnection to verify that it's configured correctly. The syntax is as follows:

```
Test-NetConnection NameOrIPAddress -Port PortNumber
```

You should run this command from somewhere other than the VM that is hosting the primary replica.

Some environments may also require that the IP address for the WSFC and selected ports (such as 445) must be accessible for administration or other purposes, which mean to configure those as part of the same or a different load balancer.

Once the load balancer is confirmed to be working, you can begin to test AG failover and connectivity to the AG via the listener.

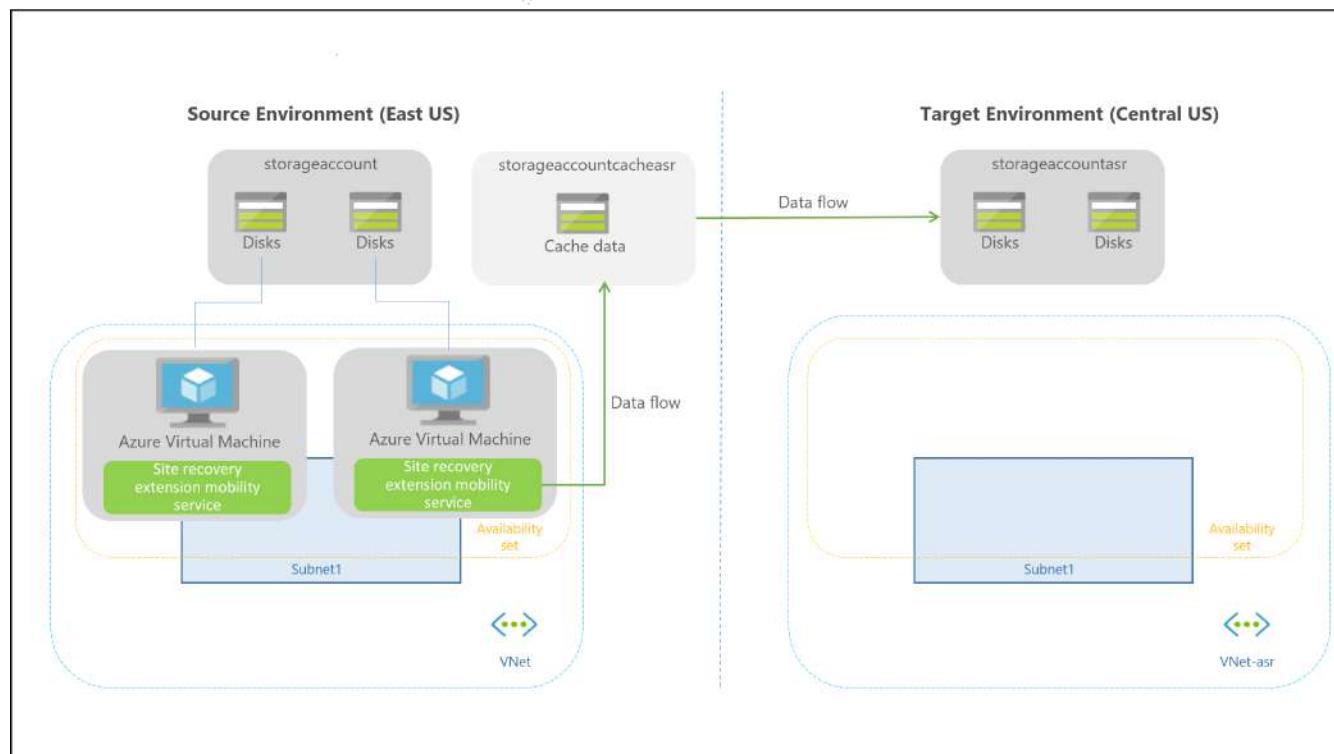
Distributed availability groups

Planning for and configuring a distributed AG is the same on premises as it is in Azure, with any Azure-specific considerations for the individual AGs. The main difference between an on-premises configuration and an Azure configuration for a distributed AG is that as part of the load balancer configuration in each region, the endpoint port for the AG needs to be added. The default port is 5022.

A traditional availability group has resources configured in a Windows Server Failover Cluster (WSFC) or if on Linux, Pacemaker. A distributed availability group does not need WSFC or Pacemaker, everything about it is maintained within SQL Server.

Azure Site Recovery

Azure Site Recovery is an option that works with the Virtual Machine, whether or not SQL Server is running inside of it. It works with SQL Server but isn't designed specifically to account for nuances that may be required when you have a specific RPO. The disks of a VM configured to use Azure Site Recovery are replicated to another region. This replication can be seen in the image below, noted by the "Data flow" arrow.



This means that all changes to a disk are replicated as soon as they occur, but this process knows nothing of database transactions. This is why recovering to a specific data point may not be possible with Azure Site Recovery in the same way it is for a SQL Server-centric solution such as when using an AG.

If it isn't possible to deploy one of the in-guest options for IaaS solutions, Azure Site Recovery is a viable option to manage disaster recovery.

Additionally, Azure Site Recovery can potentially protect you against ransomware. If infected, you could roll the VM back to a point before the infection was introduced. That could also mean data loss from a SQL Server perspective, but some data loss, especially in this case, may be more than acceptable. Up and running is often better than down for hours, days, or weeks trying to remove ransomware from your network.

The key things to know when replication is enabled on a VM:

- There's a Site Recovery Mobility extension configured on the VM.

- Changes are sent continually unless Azure Site Recovery is unconfigured or replication is disabled
- Crash consistent recovery points are generated every five minutes, and application-specific recovery points are generated according to what is configured in the replication policy.

The screenshot shows two windows side-by-side. The left window is titled 'Replication policies' and lists two policies: 'Tier1-policy-fallback' and 'Tier1-policy'. The right window is titled 'Tier1-policy' and shows its configuration details:

Source type	VMware / Physical machines
Target type	Azure
RPO threshold	15 Minutes
Recovery point retention	24 Hours
App consistent snapshot frequency	60 Minutes

Below the configuration is a section for 'Associated Configuration Servers' with one entry:

NAME	ASSOCIATION STATUS
V2AGNK-CS	Associated

For SQL Server, the **App consistent snapshot frequency** value is what you may want to adjust to reduce your RPO. However, due to the nature of how Azure Site Recovery works – it's using Volume Shadow Service (VSS) – lowering this value could potentially cause problems for SQL Server since there's a brief freeze and thaw of I/O when the snapshots are taken. The impact of the freeze and thaw could be magnified if other options such as an AG are configured. Most won't encounter issues, but if Azure Site Recovery interferes with SQL Server, you may want to consider other availability options.

If multiple VMs are part of an overall solution, they can be replicated together to create a shared crash- and application-consistent recovery points. This is known as multi-VM consistency and will affect performance. Unless VMs must be restored in this way, it's recommended not to configure this option.

One major benefit of Azure Site Recovery is that you can test disaster recovery without needing to bring down production.

A consideration for Azure Site Recovery is if there's a failover to another region, the replica VMs aren't protected when they're brought online. They'll have to be reprotected.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Describe temporal tables in Azure SQL Database

Azure SQL Database and Azure SQL Managed Instance allow you to track and analyze the changes to your data using a feature called Temporal Tables. This feature requires that the tables themselves be converted to be temporal, which means the table will have special properties and will also have a corresponding history table.

The temporal table feature allows you to use the history table to recover data that may have been deleted or updated. Recovering data from the history table is a manual process involving Transact-SQL, but could be helpful in certain scenarios such as if a user accidentally deletes important data that the business needs.

The following query searches for row versions for an employee with the filter condition WHERE EmployeeID = 1000 that were active at least for a portion of period between January 1, 2021 and January 1, 2022.

```
SELECT * FROM Employee
FOR SYSTEM_TIME
    BETWEEN '2021-01-01 00:00:00.0000000' AND '2022-01-01 00:00:00.0000000'
WHERE EmployeeID = 1000 ORDER BY validFrom;
```

Use case scenarios

The temporal table feature has various uses, including but not limited to:

- **Auditing** - Provides data audit capabilities to existing applications.
- **Historical trends (time travel)** - In time-travel scenarios, users can see how data sets changed over time.
- **Anomaly detection** - For data that don't match an expected pattern. You might investigate sales spikes that don't line up with the average, for example.
- **Data protection due to data loss** - Useful to revert undesired data change without requiring backups.
- **Slowly changing dimensions** - Dimensions in data warehousing are typically static data. However, certain scenarios require you to track data changes in dimension tables as well.

Storage consideration

Keeping historical data for a long time or performing heavy data changes may cause the history table to increase the database size more than normal tables. A large history table can increase storage cost and also affect your query performance.

It's important to establish a data retention policy within the history table, and to maintain a suitable retention policy threshold by regularly monitoring the size of the history table.

In a temporal history table, there are four ways to manage, store, and delete historical data:

- **Stretch Database** - You can configure SQL Server to silently move historical data from your temporal history tables to Azure
- **Table Partitioning** - The oldest part of the historical data can be moved out of the history table by using a sliding window approach

- **Custom Cleanup Script-** If none of the other options work, you can use a custom cleanup script to remove the data from the history table
- **Retention Policy** - A retention policy can be easily configured at the temporal table level, increasing flexibility

To explore a list of limitations for temporal table, see [Temporal Table Considerations and Limitations](#)

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document bel...

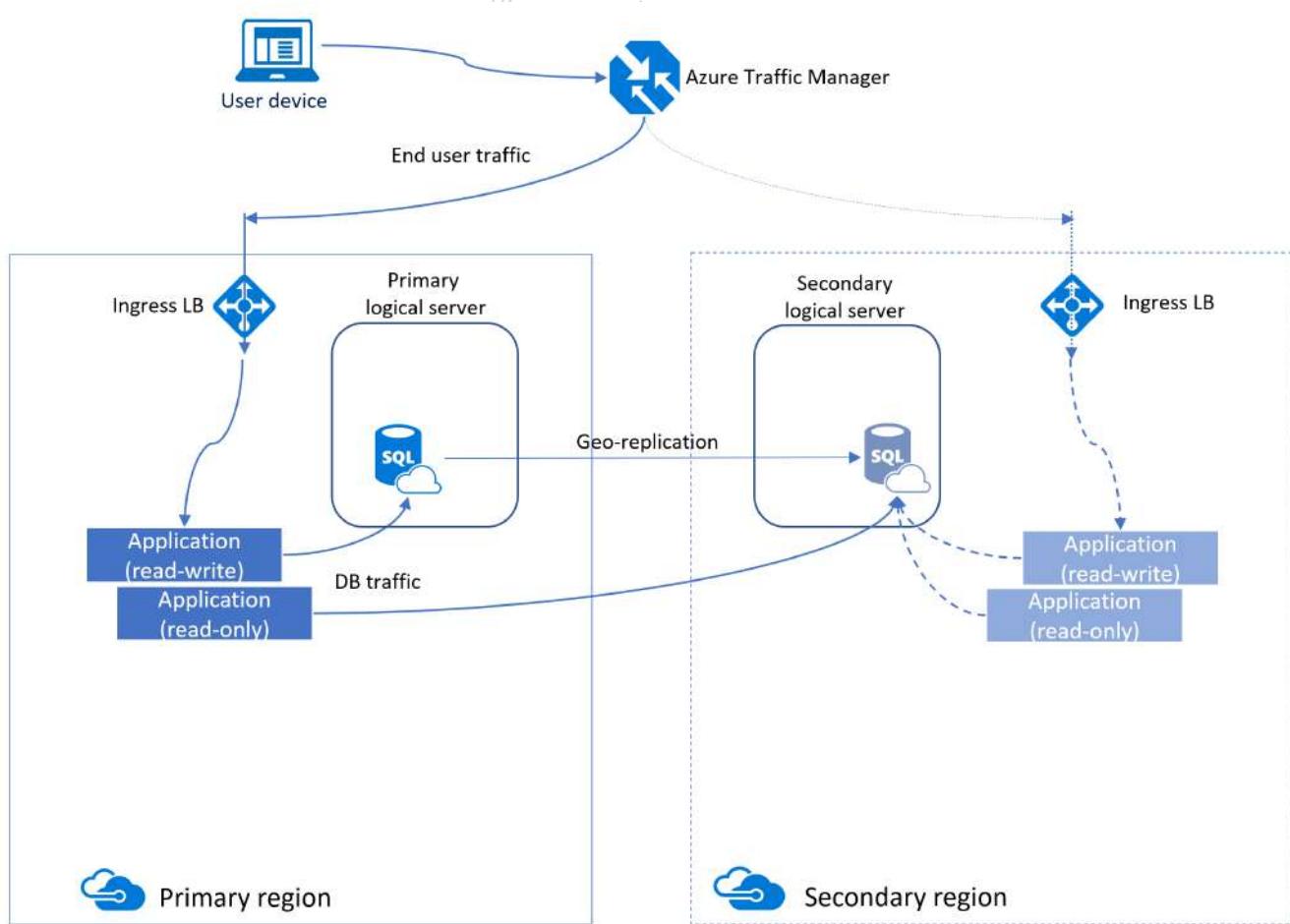
Describe active geo-replication for Azure SQL Database

One method to increase availability for Azure SQL Database is to use active geo-replication. Active geo-replication creates a secondary database replica in another region that is asynchronously kept up to date.

This replica is readable, similar to an Availability Group in IaaS. Underneath the surface, Azure uses Availability Groups to maintain this functionality, which is why some of the terminologies are similar (primary and secondary logical servers, read-only databases, etc.).

Active geo-replication provides business continuity by allowing customers to programmatically or manually failover primary databases to secondary regions during major disaster.

Azure SQL Managed Instance doesn't support active geo-replication, you must use auto-failover groups instead, which will learn on the next unit.



All the databases involved in a geo-replication relationship are required to have the same service tier.

Furthermore, in order to avoid replication overhead due to a large write workload that can affect the replication performance, it's recommended that the geo-secondary is configured with the same compute size as the primary.

As we can see above, you can manually configure geo-replication for Azure SQL Database by accessing the blade for the database, in **Data management** section, selecting **Replicas**, and then **+ Create replica**.

After the secondary replica is created, you can manually fail over your secondary replica. The roles will switch with the secondary becoming the new primary, and the old primary the secondary.

Name	Server	Region	Failover policy	Pricing tier	Replica state
AdventureWorksLT	dp300-lab-dr-23075881	South Central US	None	General Purpose Gen5, 2 vCores	Online
AdventureWorksLT	dp300-lab-dr-23075881	East US		General Purpose Gen5, 2 vCores	Readable

Cross subscription geo-replica

Some scenarios require you as a DBA to configure a secondary replica on a different subscription than the primary database. Cross subscription geo-replication is a feature that allows you to perform this task.

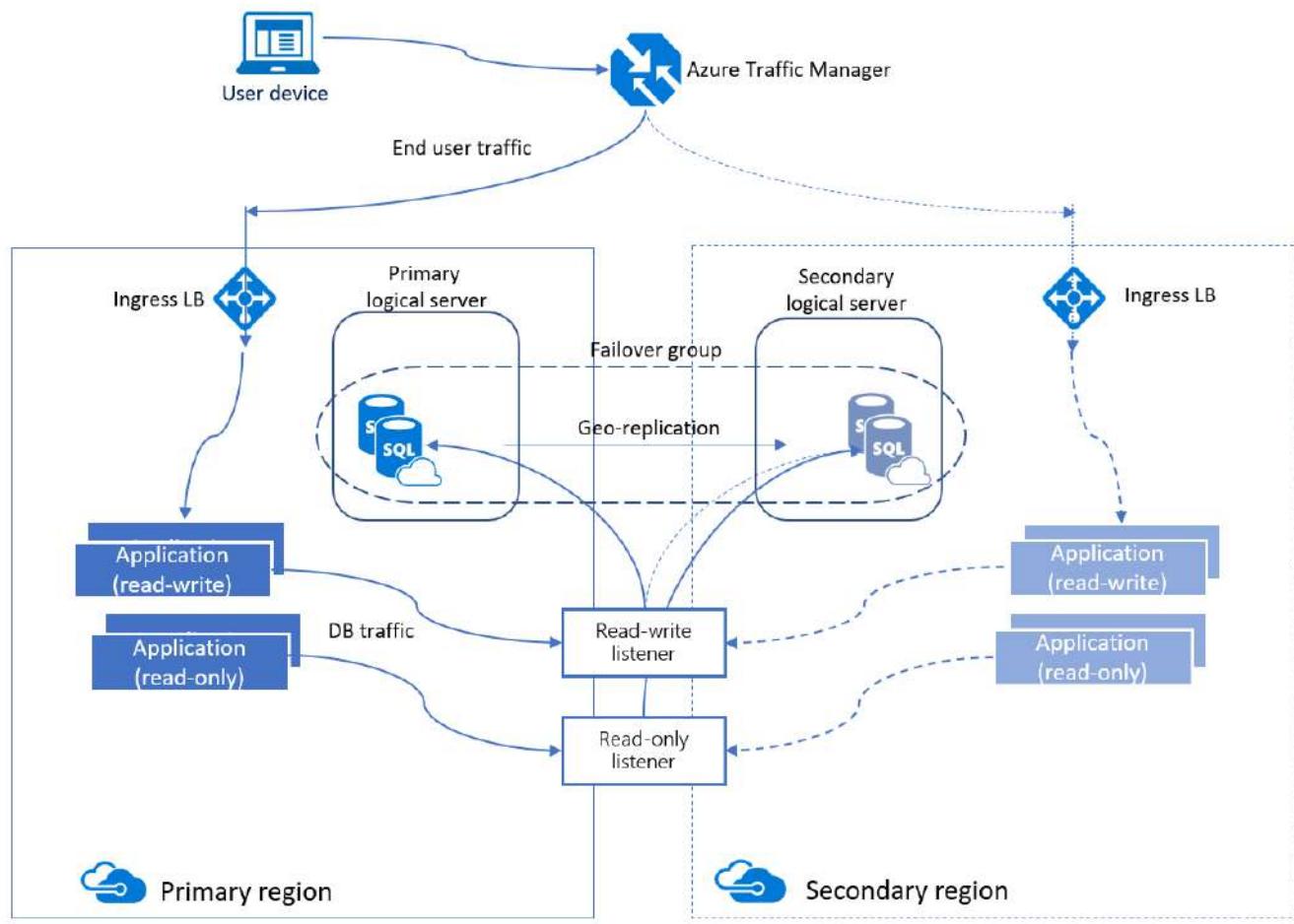
Cross subscription geo-replication is only available programmatically.

To learn more about the steps required to configure a cross subscription geo-replication, see [Cross-subscription geo-replication](#).

Explore auto-failover groups for Azure SQL Database and Azure SQL Managed Instance

An auto-failover group is an availability feature that can be used with both Azure SQL Database and Azure SQL Managed Instance. Autofailover groups let you manage how databases on an Azure SQL Database server or databases in SQL Managed Instance are replicated to another region, and let you manage how failover could happen. The name assigned to the autofailover group must be unique within the *.database.windows.net domain. SQL Managed Instance only supports one autofailover group.

Auto-failover groups provide AG-like functionality called a listener, which allows both read-write and read-only activity. This functionality can be seen in the image below which is slightly different than the one for active geo-replication. There are two different kinds of listeners: one for read-write and one for read-only traffic. Behind the scenes in a failover, DNS is updated so clients will be able to point to the abstracted listener name and not need to know anything else. The database server containing the read-write copies is the primary, and the server that is receiving the transactions from the primary is a secondary.



Auto-failover groups have two different policies that can be configured.

- **Automatic** – By default, when a failure occurs and it's determined that a failover must happen, the auto-failover group will switch regions. The ability to fail over automatically can be disabled.
- **Read-Only** – By default, if a failover occurs, the read-only listener is disabled to ensure performance of the new primary when the secondary is down. This behavior can be changed so that both types of traffic are enabled

after a failover.

Failover can be performed manually even if automatic failover is allowed. Depending on the type of failover, there could be data loss. Unplanned failover could result in data loss if forced and the secondary isn't fully synchronized with the primary. Configuring GracePeriodWithDataLossHours controls how long Azure waits before failing over. The default is one hour. If you have a tight RPO and can't afford much data loss, set the value higher. Although Azure will wait longer before failing over, this approach may result in less data loss as the secondary has more time to fully synchronize with the primary.

One auto-failover group can contain one or more databases. The database size and edition will be the same on both the primary and secondary. The database is created automatically on the secondary through a process called seeding. Depending on the size of the database, this may take some time. Ensure that you plan accordingly and that you take into account things like the speed of the network.

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

This document belongs to srinivas Ramchand Jillella.
srinivas9.dba@gmail.com
No unauthorized copies allowed!

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

What component in Azure needs to be configured for the listener in an AG to work properly?

- The NIC
- The NSG
- A load balancer

Check Answers

Multiple choice

What tool can be used to create a WSFC in Azure for AGs and FCIs?

- Wizard in Failover Cluster Manager
- PowerShell
- WMI

Check Answers

Multiple choice

What Azure feature for testing disaster recovery without bringing the production system down?

- Azure Site Recovery
- Azure SQL Database
- Azure Load Balancer

Check Answers

Multiple choice

What feature creates a replica of the database in another region that is asynchronously kept up to date?

- Failover group
- Secondary replica that is readable
- Active geo-replication

Check Answers

Multiple choice

What setting for auto-failover groups must be changed to ensure a low RPO?

- RPOZero
- AlwaysBeInSync
- GracePeriodWithDataLossHours

Check Answers

Summary

Deploying the right HADR solution in Azure is more than just picking a feature. A solution must meet your requirements, including the RTO and RPO expected by the business. Depending on the platform (IaaS or PaaS), there could be multiple options to choose from.

Depending on the PaaS option chosen for your deployment, there will be different options that you can configure to meet your availability needs, even if you don't have as many options as with an IaaS solution.

Now that you've reviewed this module, you should be able to:

- Describe what to consider when deploying a WSFC in Azure
- List what to consider when deploying an AG in Azure
- Enable AGs
- Describe Temporal Tables
- Describe active-geo replication
- Describe autofailover groups

Introduction

Even if you do nothing else to increase your availability with features like AGs or active geo-replication, you must have a solid backup and recovery strategy. If for some reason a feature fails, you'll have to restore databases from backups.

Learning objectives

In this module, you'll learn:

- Database backup and restore options for IaaS
- Virtual machine backup and restore options for IaaS
- Backup and restore options for PaaS

Back up and restore SQL Server running on Azure virtual machines

SQL Server has two types of databases: system and user. System databases are the ones used by SQL Server such as master and msdb. User databases are the ones created by users that store the data for applications. Both are important to account for when devising a backup and recovery plan. The nature of most system databases is that they're updated less frequently, although there are exceptions. As a general rule of thumb, system databases aren't restored from one SQL Server instance to another. Your main concern should be backing up the user databases.

The most common types of backups generated for SQL Server installations are full, differential, and transaction log. Depending on the deployment method, not all of these may be available as an option.

A full database backup is a backup of a single database. When the backup is made, all the pages from the database are copied to the backup device. The backup contains enough information so that you can restore the database to the point at which the backup was made. If you want to restore to a specific point-in-time to achieve your Recovery Point Objective (RPO), that can happen with the use of differential and/or transaction log backups. A full database backup backs up all the changes made to the database by the time the backup finishes.

A differential backup contains all the database pages that have changed since the last time a full backup was made.

A transaction log backup isn't only used to be able to achieve RPO and get to a more granular point in time but clears the transaction log and keeps its size manageable. Transaction log backups can be generated as frequently as every 30 seconds, although that is impractical.

IMPORTANT: Understand how the transaction log works because it impacts not only how the transaction log is backed up, but also how you can do point-in-time recovery using transaction log files.

There are other backup options such as copy-only, file, filegroup, partial, and more.

A differential or a log backup can be restored after a full database is restored, as long as the database RESTORE command uses either the `WITH NORECOVERY` or the `WITH STANDBY` option. If neither option is used, the database RESTORE will do a recovery of the database, after which no extra backups can be applied.

Every SQL Server database uses one of three recovery models: FULL, BULK_LOGGED, or SIMPLE. The recovery model is set as a database option, and governs the type of backups and restores that could be used with the database. Most databases are set to FULL or SIMPLE. FULL allows all types of backups to be generated while SIMPLE doesn't allow transaction log backups. This means that if you have a smaller RPO, SIMPLE may not meet your needs as you can't restore to a granular point in time.

Back up a SQL Server virtual machine

Azure Backup can back up VMs that contain SQL Server. These backups would contain not just SQL Server databases; they would everything that is in the VM so it could be restored as a whole. While this option may not be right for everyone, it can potentially protect against problems like ransomware.

VM-level backups are SQL Server-aware, also known as application aware, so they'll create an application-consistent backup. This means that if you restore a VM-level backup, it will not 'break' SQL Server. If using this option, when looking in the SQL Server log you'll see that the I/O has been momentarily frozen and then starts again when complete. If this causes issues with availability features like AGs, you may want to consider another backup strategy.

Combining SQL Server backups with snapshots can potentially cause issues. If snapshot delays cause backup failures, set following registry key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\BCDRAGENT]
```

```
"USEVSSCOPYBACKUP"="TRUE"
```

Use local disks or a network share for backup files

As with on premises SQL Server instances, databases can be backed up to disks attached to the VM or to network shares (including the file share in Azure called Azure Files) that SQL Server has access to. If you're backing up to disks local to the VM, ensure that they aren't written to the ephemeral storage that is erased upon shutdown or restart. You might also want to make sure that the backups are copied to a second location so as not to create a single point of failure.

Backup databases to and restore from URL

Another option is to configure backup to URL for the SQL Server instance installed in the VM. Unlike backups made on premises, backup and restore from URL for an IaaS VM is effectively a local option.

Backup to URL requires an Azure storage account and uses the Azure blob storage service. Inside the storage account, there are containers, and the blobs are stored there. Unlike a path on your local disk, the path to a backup file will look something like

`https://ACCOUNTNAME.blob.core.windows.net/ContainerName/MyDatabase.bak`. You can include more folder names under your container for easier identification of backups (for example, FULL, DIFF, LOG).

To backup to or restore from a URL, authentication must be established between the SQL Server instance and Azure. Remember that inside of an Azure VM, SQL Server doesn't know it's running on Azure. A SQL Server Credential can be composed of the Azure storage account name and access key authentication or a Shared Access Signature. If the former is used, the backup will be stored as a page blob and if the latter, it will be stored as a block blob. Starting with SQL Server 2016, only block blob is available so you should use a Shared Access Signature. From a cost perspective, block blobs are also cheaper, and Shared Access Signature tokens offer better security control.

Restoring from a URL is as simple as restoring from disk or a network share. In the SQL Server Management Studio UI, select URL from the backup media type in the Wizard. If using Transact-SQL, instead of using FROM DISK, you would use FROM URL with the appropriate location and backup file name(s). Here are some sample statements:

The following statement would back up a transaction log.

```
BACKUP LOG contoso  
TO URL = 'https://myacc.blob.core.windows.net/mycontainer/contoso202003271200.trn'
```

The following statement would restore a full database backup without recovering it, so that a differential or transaction log backups could be applied.

```
RESTORE DATABASE contoso  
FROM URL = 'https://myacc.blob.core.windows.net/mycontainer/contoso20200327.bak'  
WITH NORECOVERY
```

Automated backups using the SQL Server resource provider

Any IaaS VM that has SQL Server installed can use the SQL Server resource provider. One of its options is the ability to configure automated backups so Azure takes care of backing up SQL Server databases. It requires the use of a storage account.

One benefit of implementing backups this way is that you can manage retention times for the backups. Another benefit is that you can ensure RPO due to the ability to take database and transaction log backups all in one easy-to-configure place. The image below shows an example of what configuring an automated backup looks like in the Azure portal.



AUTOMATED BACKUP

Configure backups for databases in your virtual machine. All your SQL Server databases in this virtual machine will be backed up automatically per the settings you choose. If you decide to change settings via SQL Server Managed Backup in the future, the new settings will override the Automated Backup settings.

Automated backup

Retention period (days)

30

Storage account *

module7storage (Standard_RAGRS)

<https://module7storage.blob.core.windows.net/>

[Select storage account](#)

Encryption

Backup system databases ⓘ

Configure backup schedule

Specify the schedule for full and log backups

FULL BACKUP SCHEDULE

Backup frequency

Take full backups every day at the specified start time

Backup start time (local VM time)

Full backup time window (hours)

1

LOG BACKUP SCHEDULE

Backup frequency (minutes)

10

All your SQL Server databases in this virtual machine will be backed up automatically per the settings you choose. If you decide to change settings via SQL Server Managed Backup in the future, the new settings will override the Automated Backup settings.

The automated backup option is currently only available for Windows Server-based SQL Server installations.

IMPORTANT: You choose one method of backing up databases with IaaS-based SQL Server deployments. For example, if you use automated backups, especially with transaction log backups, do not also configure those at the instance level inside the VM. You could cause problems with the log chain with restoring a database if things are uncoordinated, because each log backup clears the log and you must have an entire unbroken chain of log backups in order to do a log restore. For example, if transaction log backups happen inside the guest as well as at the Azure level, you may have to piece together the backups to do a restore.

While the backups can be automated, restores can't be. You would need to configure and use the restore from URL functionality within SQL Server.

Back up and restore a database using Azure SQL Database

Back up and restore on SQL Server PaaS offering work differently than on IaaS. Backups are generated automatically for Azure SQL Database, and Azure SQL Managed Instance. A full backup is created once a week, a differential every 12 hours, and transaction log backups every 5 – 10 minutes. All backups are located in read-access, geo-redundant (RA-GRS) blobs replicated to a datacenter that is paired based on Azure rules. That means backups are safe from an outage in a single data center.

Database backup and restore for SQL Database

SQL Database can assist you to be compliant with mandatory backups for regulatory purposes with retention policies. Backup policies can be configured per database as shown in the image below:

The screenshot shows the 'Configure policies' section for an Azure SQL Database. It includes fields for PiTR backup retention (set to 0 days), weekly LTR backup settings (unchecked), monthly LTR backup settings (unchecked), and yearly LTR backup settings (unchecked). Each setting has a dropdown for selecting the retention period in weeks.

Configure policies
SQL server

Point In Time Restore Configuration

Configure PiTR backup retention Days

Long-term Retention Configurations

Weekly LTR Backups ⓘ

How long would you like weekly backups to be kept?

0 Week(s)

Monthly LTR Backups ⓘ

How long would you like the first backup of each month to be kept?

0 Week(s)

Yearly LTR Backups ⓘ

Which weekly backup of the year would you like to retain?

Week 1

How long would you like this annual backup to be kept?

0 Week(s)

If the server containing the database is deleted, all backups will be deleted at the same time, and there's no way to recover them. If the server isn't deleted but the database is, you can restore the database normally.

Both SQL Database, and SQL Managed Instance have a feature called Accelerated Database Recovery (ADR). This feature is enabled by default, and its purpose is to decrease the time it takes to deal with long running transactions so they don't affect the recovery time. Although Accelerated Database Recovery was developed for Azure and was originally an Azure-based feature, ADR was implemented in SQL Server 2019 as well.

NOTE: You can't restore SQL Database Managed Instance backups on SQL Database.

Point in time restore

To restore a database to a specific point in time on SQL Database, you can use either Azure portal, Azure PowerShell, Azure CLI, or REST API.

Home > [SQL databases](#) > [AdventureWorks \(contoso/AdventureWorks\)](#) > [contoso](#) > [AdventureWorks \(contoso/AdventureWorks\)](#)

Create SQL Database - Restore database

Microsoft

[Basics](#) [Review + create](#)



Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription [\(i\)](#)

My Subscription [\(i\)](#)

Resource group [\(i\)](#)

contoso_rg [\(i\)](#)

Source Details

Select a backup source and details. Additional settings will be defaulted where possible based on the backup selected.

Source Database

AdventureWorks

Select source

Point-in-time [\(i\)](#)

Earliest restore point

2021-11-23 00:00 UTC

Restore point (UTC) *

11/30/2021 [\(i\)](#) 1:51:00 PM

Choose a restore point between the earliest restore point and the current time in UTC.

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name *

AdventureWorks_2021-11-30T13-51Z

Server [\(i\)](#)

contoso (West US) [\(i\)](#)

Want to use SQL elastic pool? * [\(i\)](#)

Yes No

Compute + storage * [\(i\)](#)

General Purpose

Serverless, Gen5, 2 vCores, 32 GB storage

[Configure database](#)

The image above shows the SQL Database restore page on Azure portal, where you can restore a database to a specific point in time.

Restore in place isn't supported on SQL Database, and SQL Managed Instance. You need to make sure the database doesn't exist before attempting the restore operation. By default, point in time retention policy is set to

seven days, and you can change it to up to 35-days.

Restore a deleted database

Both SQL Database, and SQL Managed Instance have a feature to restore a deleted database to the last point in time available before the `DROP DATABASE` took place.

Database	Deletion time (UTC)	Creation time (UTC)	Edition
MyDatabase	2021-11-16 20:10	2021-11-16 19:54	

The image above shows how to restore a deleted database on SQL Database. The *deleted databases* page shows a list of deleted databases available to restore, the database deletion time in UTC, and the database creation time in UTC. Once you select the database, the *Create SQL Database - Restore database* page will open. On that page you'll find the earliest restore point in time available for the selected database.

Database backup and restore for SQL Managed Instance

Azure manages backups for databases in SQL Managed Instance automatically, and they operate similar to SQL Database.

You can also manually back up, and restore databases with SQL Managed Instance using the same backup to URL/restore from URL functionality found in SQL Server covered earlier. That requires the use of credentials to access the Azure Blob Storage container. SQL Database doesn't support this feature.

You can only generate a `COPY_ONLY` backup since SQL Managed Instance is maintaining the log chain. A sample backup statement would look like:

```
BACKUP DATABASE contoso
TO URL = 'https://myacc.blob.core.windows.net/mycontainer/contoso.bak'
WITH COPY_ONLY
```

NOTE: You can't restore SQL Database Managed Instance backups on SQL Database.

Knowledge check

Choose the best response for each of the questions below. Then select **Check your answers**.

Multiple choice

How does backup to URL in SQL Server or Azure Managed Instance store the backup file?

- As a URL
- As a blob
- As a pointer

Check Answers

Multiple choice

What is the default redundancy option for Azure SQL databases?

- Locally-redundant backup storage (LRS)
- Zone-redundant backup storage (ZRS)
- Geo-redundant backup storage

Check Answers

Multiple choice

Which option should you use to geographic failover instances of SQL Managed Instance?

- Auto-failover groups
- Active geo-replication
- Failover Cluster Instance

Check Answers

Summary

Without backups, you don't have high availability or disaster recovery. Features like AGs or active-geo replication don't replace a proper backup and recovery strategy. Backups must be tested via a restore otherwise your backups are just files that sit on a file system that checks a box somewhere that backups were generated.

Now that you've reviewed the module, you should be able to describe:

- Database backup and restore options for IaaS
- Virtual machine backup and restore options for IaaS
- Backup and restore options for PaaS