AZURE SQL DB

Design, Implement and Manage PaaS SQL Server Databases

Martin Cairney

Lead Consultant with readify
A Telstra Company

Microsoft Data Platform MVP
Microsoft Certified Trainer

Organiser – SQL Saturday Melbourne
(https://bit.ly/2T4SB6S)

@martin_cairney

http://au.linkedin.com/in/martincairney

MVP Microsoft Most Valuable Professional

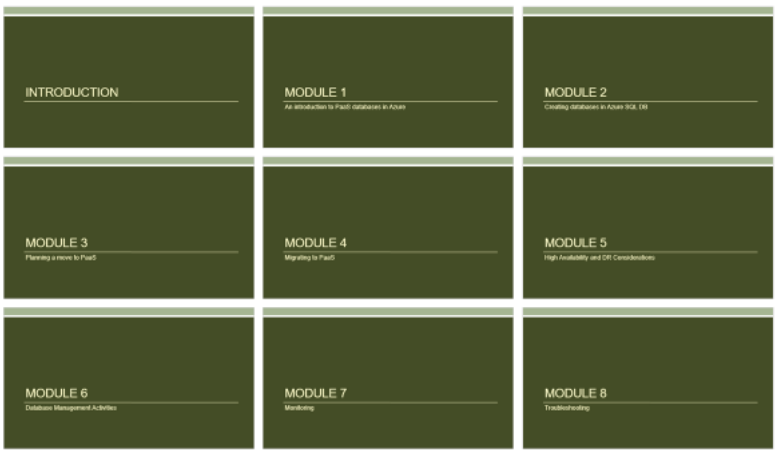
PASS LOCAL GROUP LEADER

PASS OUTSTANDING PASS VOLUNTEER

Microsoft Certified Trainer — MCT — 2019 - 2020

Microsoft Certified Professional — MCSE — Data Management and Analytics — Earned 2017

Agenda



INTRODUCTION

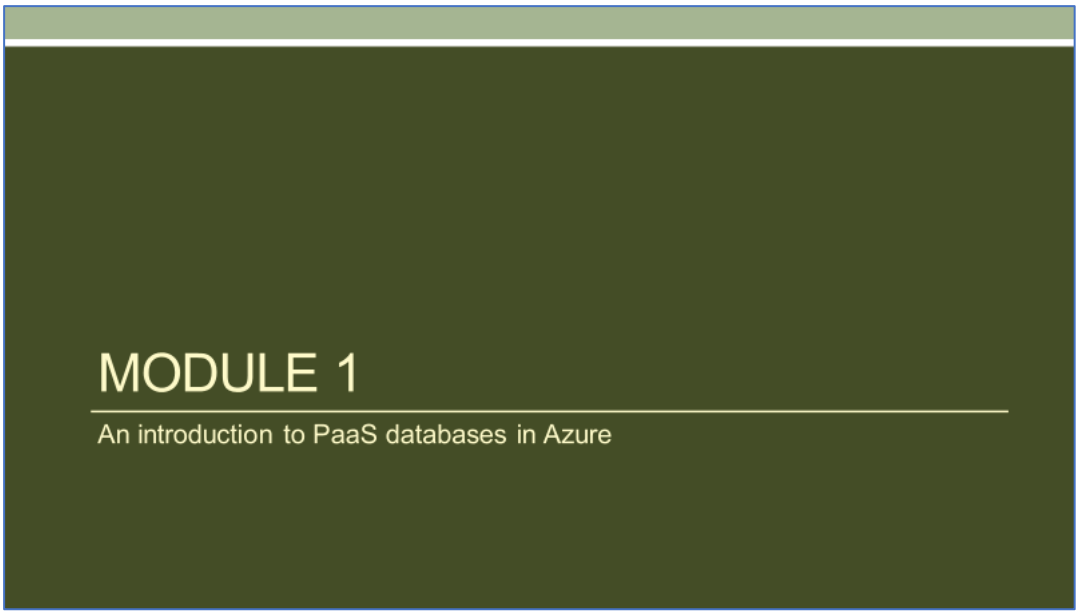
Pre-Requisites

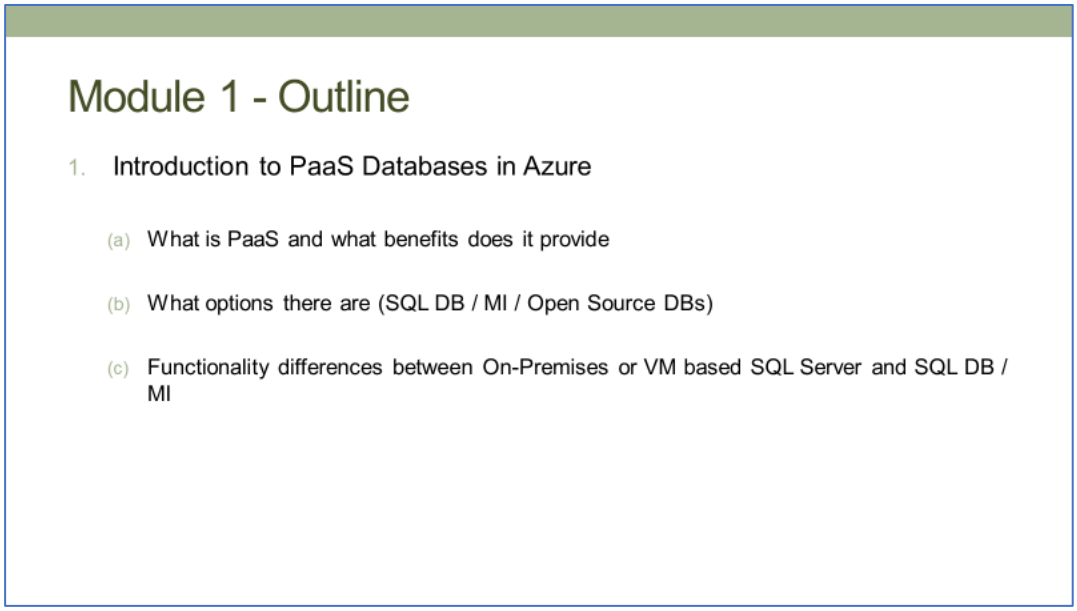
You'll need the following for this course:

- Access to an Azure subscription with permissions to create resources
- Azure PowerShell installed on your computer
- Access to an "on-premises" instance of SQL Server with a copy of AdventureWorks. Your login should be a member of the sysadmin role
- Slack installed on your computer and access to a custom training channel

Baseline Skills

- SQL Server Database Administration
  - Backup and recovery
  - High Availability and Disaster Recovery
  - Extended Events
  - Auditing
  - Alerts
  - Encryption
- PowerShell





Introduction

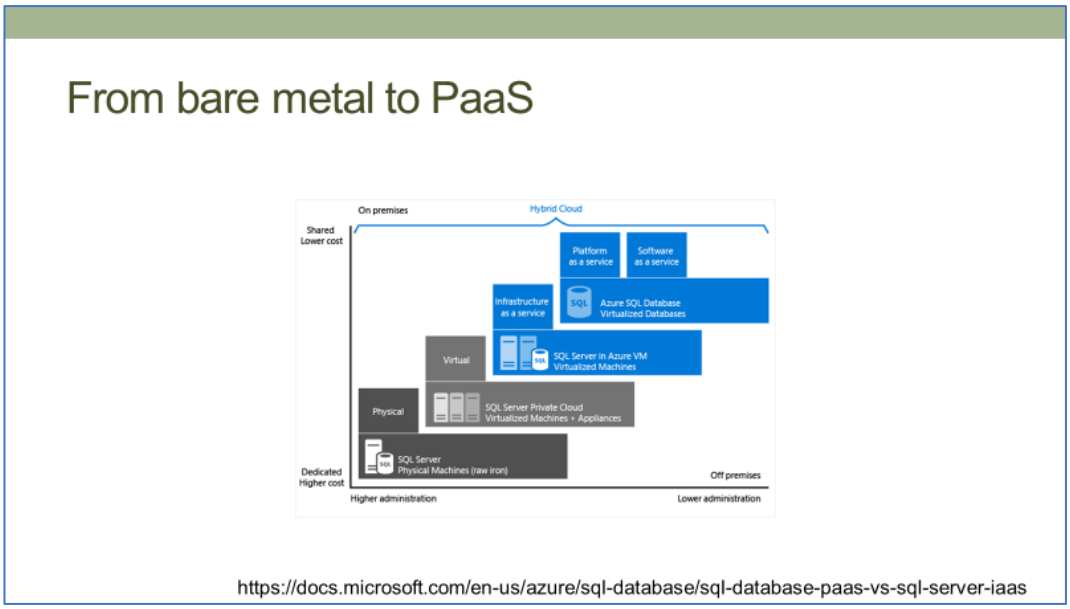
Before we can start to work with PaaS databases, we need to understand what they are and how they have come to be and are positioned in the world of virtualisation.

In this module we will look at the progression from SQL Server on bare metal to PaaS along with some of the real-world benefits that some customers have achieved by migrating their databases to PaaS.

We will then look at the various PaaS Database services in Azure and get an overview on what they provide compared to their on-premises equivalents.

Resources

Azure SQL Database Documentation:	https://docs.microsoft.com/enus/azure/sql-database/
Azure Updates:	https://azure.microsoft.com/enus/updates/
Azure DB for MySQL Documentation:	https://docs.microsoft.com/enus/azure/mysql/
Azure DB for MariaDB Documentation:	https://docs.microsoft.com/enus/azure/mariadb/
Azure DB for PostgreSQL Documentation:	https://docs.microsoft.com/enus/azure/postgresql/
Choose a version of Azure SQL:	https://docs.microsoft.com/enus/azure/sql-database/sql-database-paas-vs-sql-server-iaas
Feature Comparisons:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-features
SLA Guarantees:	https://azure.microsoft.com/en-au/support/legal/sla/sql-database



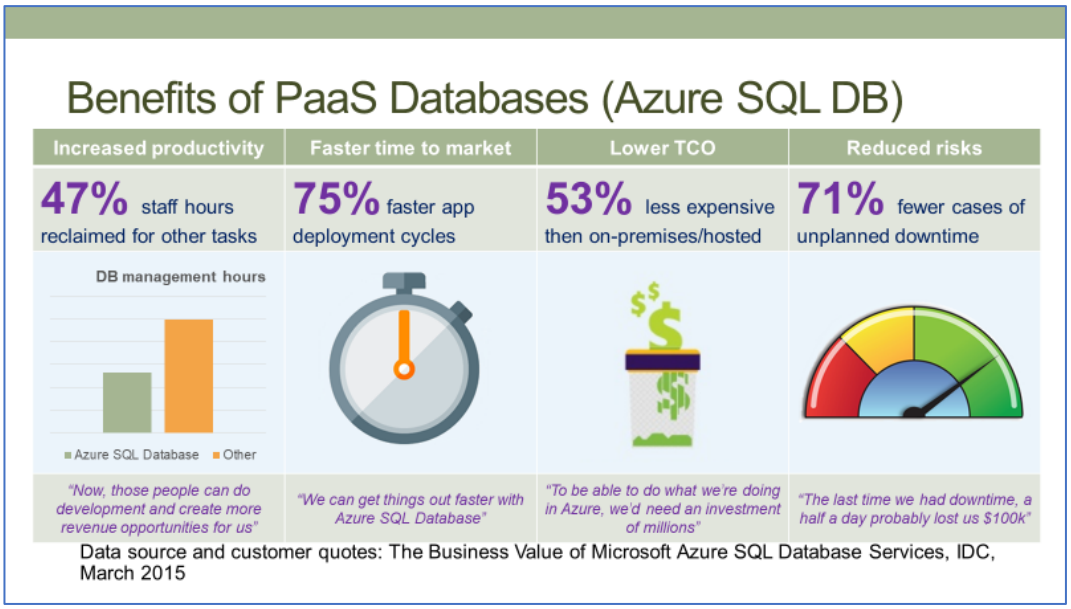
Virtualisation and abstraction has evolved over the years. Starting from the days of raw iron deployments of SQL Server on dedicated physical machines which required administration at a number of levels – Storage Admins, Network Admins, Server Admins and Database Admins. The dedicated physical location in a Data Centre, the power consumption and the multiple levels of administration meant higher costs.

Virtualisation technologies such as Hyper-V and VMware helped to make some reduction in costs by sharing the physical infrastructure among many virtual servers. There was still additional administration effort needed but as it was virtualised, the ongoing administration was more often solely in the hands of the VM Admin (storage, networking and the physical server clusters only touched rarely).

Public clouds opened the way for the full stack to be defined in code. The underlying physical platform is designed and operated independently from any IaaS deployment on top of it and effectively all of the physical administration is removed. Defining the VM configuration in code allows for more reuse in the way of templates so deployments become easier and quicker. The Database Administrator role however is still much the same.

Finally at the PaaS/SaaS level, even the VM definition has been absorbed into the underlying cloud provider service and left the database as the final layer to be configured as needed for the applications. PaaS databases are often trumpeted as requiring almost no Database Administration involvement, however this is not really the case as we will learn throughout this course. Moving to PaaS can result in reduced costs but and unmanaged database can creep up in costs and administration overhead unless the DBA focus is on continued cost reduction.

So, if your main intention is to move to Azure SQL DB to lower your administration efforts and cost, you will almost certainly be able to reduce your overall costs, but your administrative efforts (if you want to ensure that you have minimised your costs as much as possible) won't go away immediately. As you move to PaaS databases, your DBAs will need to start moving away from the infrastructure side of things and focus more on the performance and management aspects in more detail to cut, and maintain the reduction in the costs.



Question for the Attendees - Why would you want to move to Azure SQL DB or a PaaS offering?

Write down your top 3 reasons for wanting to move to Azure SQL DB

DB MANAGEMENT HOURS:

This is where you can have an impact – if you build and automate it well then your time spend will drop possibly even more than the 47% claimed here.

FASTER APP DEPLOYMENT CYCLES:

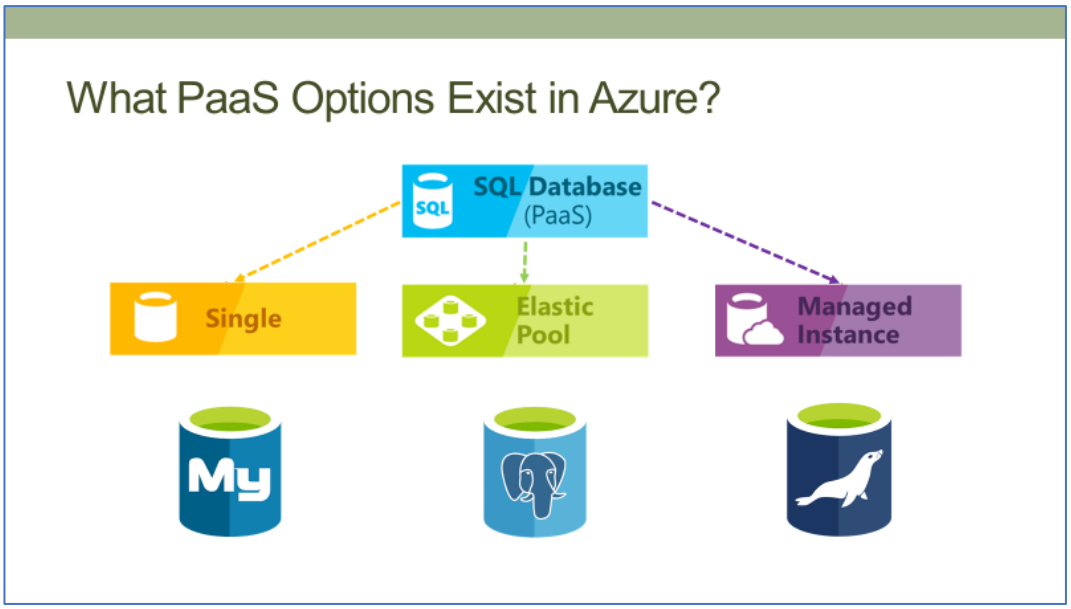
As there is less to build out then you can have you different environments all provisioned in seconds.

LESS EXPENSIVE THAN HOSTED:

You need to add up all the costs for all the components you need to have a hosted SQL Server instance – the VM, the hardware, the OS patching and finally the DBA effort all rolled into a monthly cost

UNPLANNED DOWNTIME:

This is effectively realized by the financial SLAs provided by Microsoft – so we get this straight out of the box so to speak



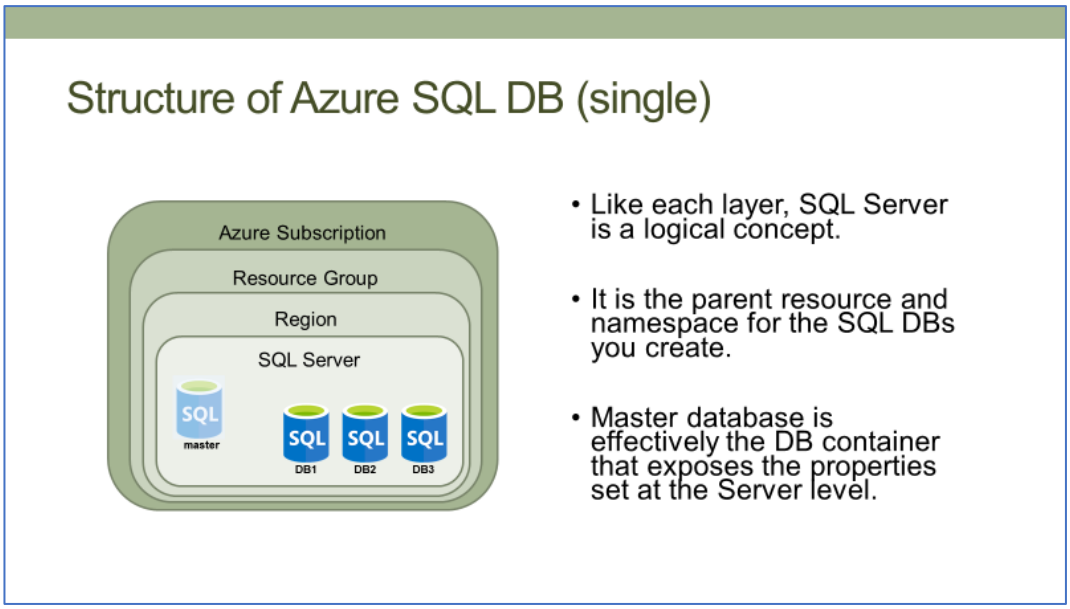
Within the PaaS SQL Server family, first there was just a single database. Although this was contained in a logical server, the server is not a real physical concept and so the database can be considered self-contained. With the performance tier, this could be considered similar to Resource Governor at the on-premises version setting the restrictions per database.

The release of the various "Elastic" tooling components opened up more configurations to interact across databases. The pool was also configured as the performance tier governor – meaning that individual databases could consume up to the entire pool resources at any time (if there was no other demands from the other databases in the pool). This would be like the Resource Governor at the instance level and with some constraints at the database levels.

The latest component is Managed Instances. Here you effectively get the full power of the underlying VM to use but without the overhead of managing the VM. You still get many of the PaaS benefits but as you are not sharing with any neighbours you can use more of the functionality from the on-premises version that was not previously available with the other offerings.

And, there is more …..

So there is now not only SQL Server as a PaaS offering. Azure DB for MySQL and Azure DB for PostgreSQL are GA and Azure DB for MariaDB is in Preview.



As for all other Azure services, there are a number of layers involved in creating an Azure SQL DB.

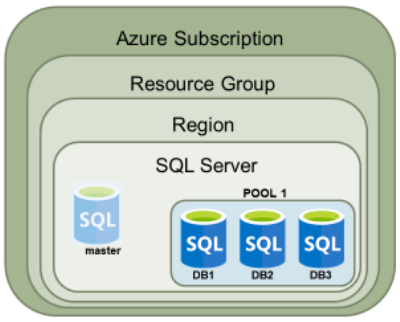
To create the database, you first need to have a subscription and within that subscription create a resource group. Within the resource group you select the region and then create a SQL Server. This does not create a server in the same way as an instance of SQL Server on-premises but sets the core properties such as the namespace used to connect, the administrators, a firewall (since the IP Address is by default an internet facing IP Address – there is scope now to restrict access to VNETs in Azure as well).

Some functionality can be switched on at the server level which means that it is in place for any database that is created within this server namespace. This covers logins, firewall rules, auditing rules, threat detection policies, and failover groups.

Note that the master database is not exposed in the portal – most of the features and functionality within the master database are exposed in other ways – e.g. the server admin which is displayed as a user in master when connected using SSMS.

You can add multiple Azure SQL DBs to the same server namespace – each of these are self-contained (with the exception being that logins created in master can have access to all databases in that server namespace). They can have different Performance Tier levels. There is also no guarantee that all resources (the server namespace and the databases) will be physically in the same location in the region's data centre.

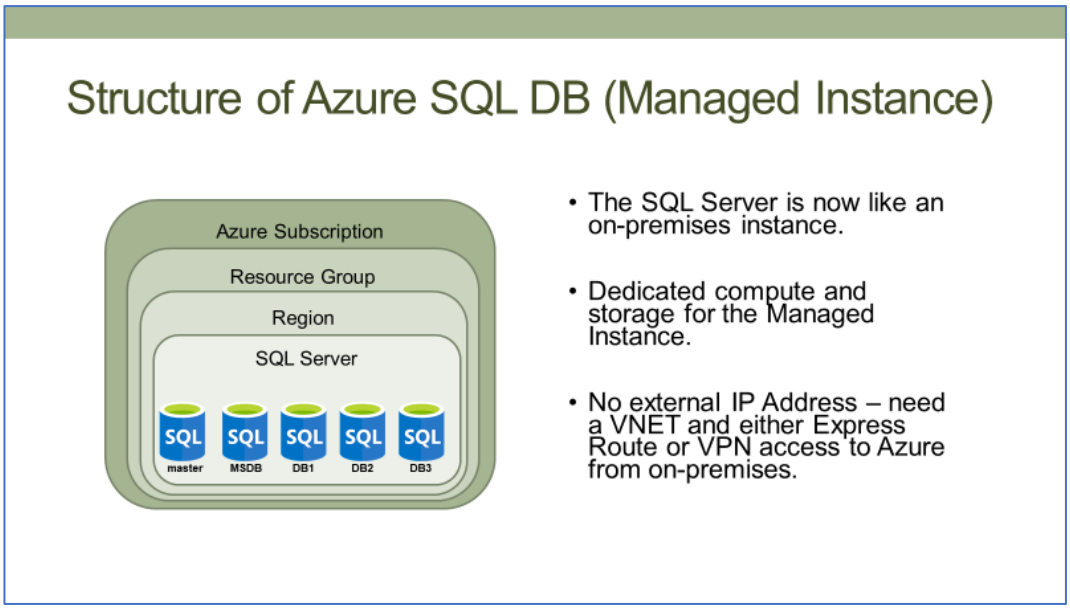
Structure of Azure SQL DB (Elastic Pool)



- The Server structure is the same as for the single database version.
- The individual databases are further constrained by the pool resources.
- Best suited for sets of databases with low average consumption with relatively infrequent spikes.

The same server namespace that we used for the single database can also be used by an Elastic Pool (or multiple Elastic Pools).

When we create an Elastic Pool we have a set of resources that are available across all of the databases. Each database can also be further restricted to a common maximum and minimum set of resources.

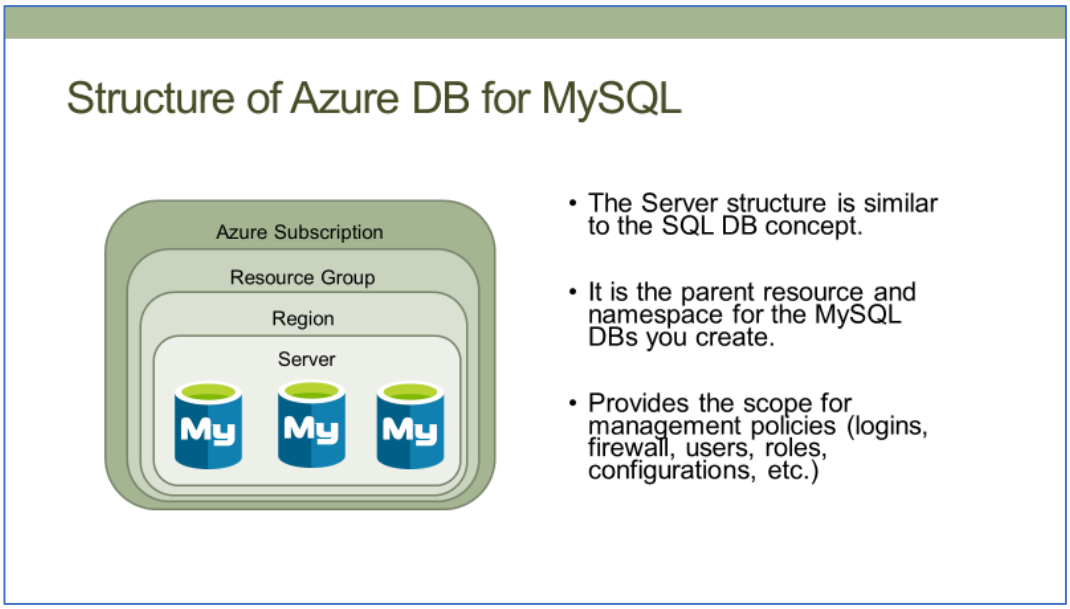


A Managed Instance has its own dedicated compute and storage under the covers so you effectively have something like a VM to yourself rather than shared as in the other PaaS offerings.

However, there are a number of features in common with the single DB or Elastic Pool of DBs such as the automated backups to ensure point-in-time recovery. Users can also run their own native backups provided that they use the COPY_ONLY option.

There are some features that are more restrictive than the single DB or Elastic Pool of DBs such as the need to have an internal Azure VNET - no external IP Address access. This means that your client applications need to have access to that VNET – and the same for your SSMS installation.

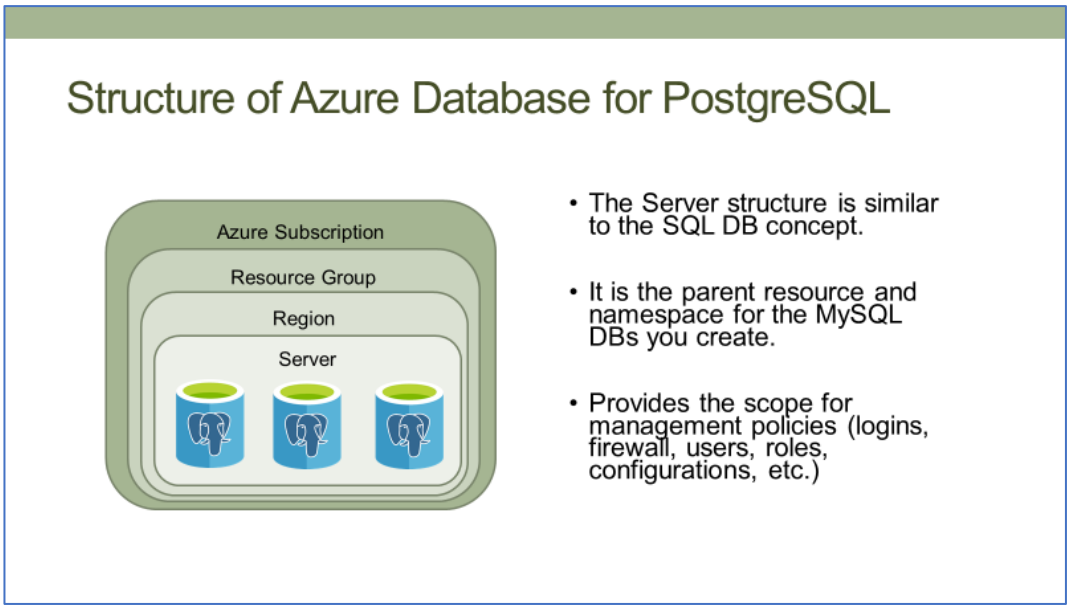
Will be very close to feature compatible with on-premises SQL Server 2017 by GA.



It's a similar concept to Azure SQL DB – the server is the connection namespace and the container for the firewall, logins, configurations etc..

Azure DB for MySQL is based on the MySQL Community Edition engine.

One main difference compared to Azure SQL DB is that you can select the version of Azure DB for MySQL that you want to install - currently 5.6 or 5.7 based on the InnoDB engine and version 8 currently in preview.

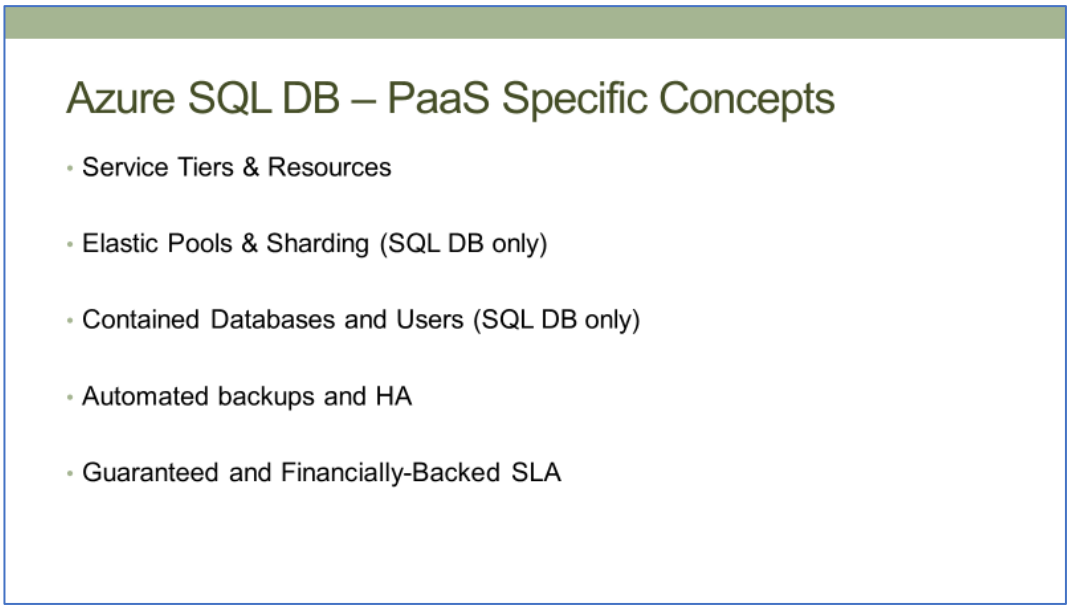


Once again it's a similar concept to Azure SQL DB – the server is the connection namespace and the container for the firewall, logins, configurations etc..

Azure DB for PostgreSQL will support up to version n-2 of the PostgreSQL engine. This provides some flexibility on the version of PostgreSQL that you want to install - currently versions 9.5, 9.6, 10.7 and 11.2.

There is also support for many of the PostgreSQL Extensions in the following groups:

- Data Types Extensions
- Functions Extensions
- Full-Text Search Extensions
- Index Types Extensions
- Language Extensions
- PostGIS Extensions
- Miscellaneous Extensions



The main aspects about Azure SQL DB that are different to on-premises that you need to be aware of are as shown

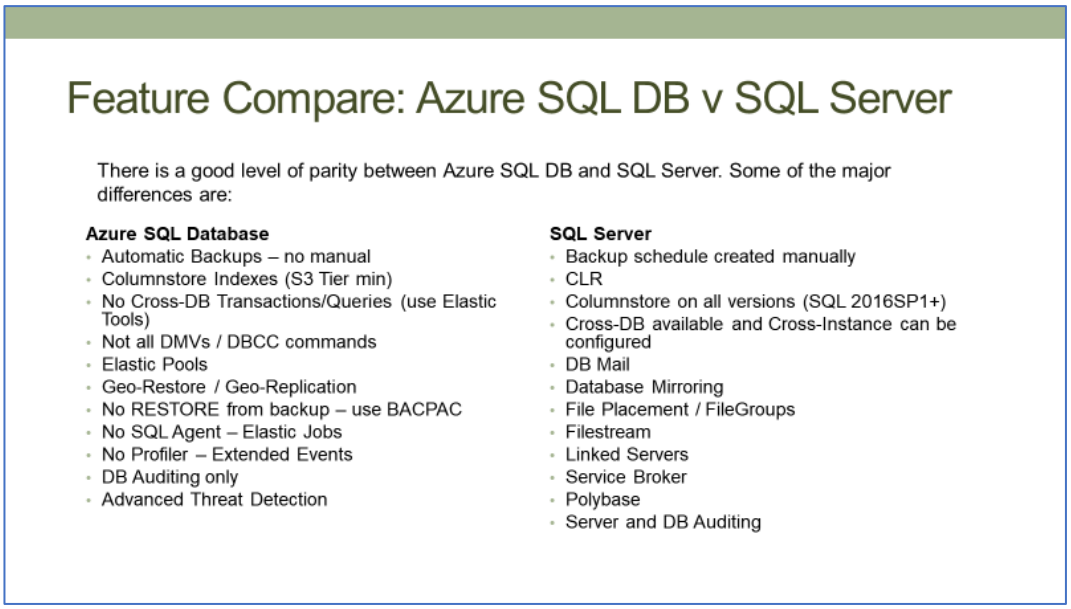
Service Tiers & Resources : provides the method for scaling resources and also still enforces some function differences (e.g. columnstore indexes only in S3 and above Tiers).

Elastic Pools & Sharding : either a method to share resources of DBs with intermittent performance needs or to enable a scale out architecture with the shards being allocated to the same or different Elastic Pools. The Elastic toolset includes Elastic Query, Elastic Jobs, Elastic Transactions.

Contained Databases and Users : the database is a boundary – no cross DB unless using the Elastic tools. Also the users are fully contained in the database and provide the security context of access to a single DB only.

Automated backups and HA : backups are taken automatically with retention between 7 and 35 days depending on the tier. Also multiple copies kept locally for redundancy and the option for Geo-Replication to provide up to 4 readable secondaries in different regions. Committed Transactions are replicated asynchronously for performance.

Guaranteed and Financially-Backed SLA : All Service Tiers have an uptime SLA of at least 99.9%. With the different configurations available, this can be increased up to 99.995% when Zone Redundant deployments are used.

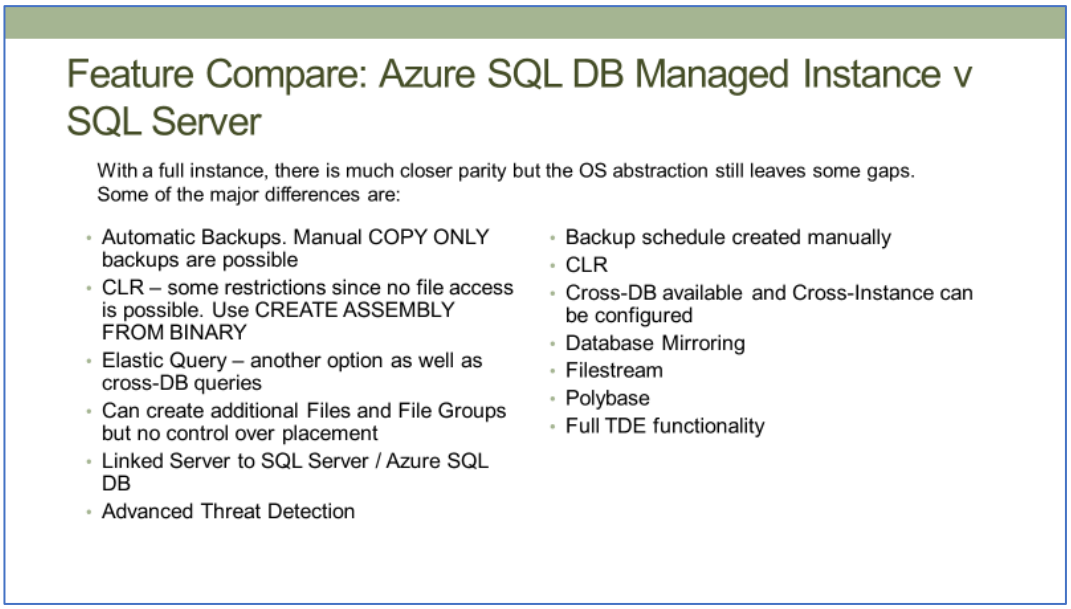


SQL DB aims to be as close as possible to the on-premises version (makes migrations between versions simpler)

However, as there is no access to the OS and no custom installations and a shared platform underpinning it then there is a restriction on some functionality.

The areas that are missing or different include:

BACKUP/RESTORE - this is done automatically
CLR access – needs OS level access which could impact other tenants
DBMail / Event Notifications – and other items that use MSDB on-premises
DTC – Elastic Transactions may cover some of this
File Groups / Filestream – no control over file placement or direct storage access
Linked Servers - We can use Elastic Query instead
SQL Agent – Elastic Database Jobs are similar
Profiler - Extended Events How many of you have moved to Extended Events?



Azure SQL DB Managed Instance aims to deliver close to 100% surface area compatibility with on-premises SQL Server (Enterprise Edition) coming in stages until service general availability.

Managed Instance allows existing SQL Server customers to lift and shift their on-premises applications to the cloud with minimal application and database changes while at the same time, preserving the PaaS capabilities (automatic patching and version updates, automated backups, high-availability).

There is much more functionality available but still restrictions apply to anything that may be looking for OS or disk access.

One major advantage with Managed Instances is that if you lift and shift existing SQL Server 2008 or SQL Server 2008 R2 instances then you may be able to continue extended support for a further period of time.

General Azure SQL DB Service Restrictions

Maximum	Logical server	Managed instance
Databases per server/instance	5000	100
Default number of servers per subscription in any region	20	N/A
Max number of servers per subscription in any region	200	960 for EAs (1440 in some regions)
DTU / eDTU quota per server	54,000	N/A
vCore quota per server/instance	540	80
Max pools per server	Limited by number of DTUs or vCores	N/A

What happens when you get close to these restrictions?

As the number of databases approaches the limit per server, the following can occur:

- Increasing latency in running queries against the master database. This includes views of resource utilization statistics such as sys.resource_stats.
- Increasing latency in management operations and rendering portal viewpoints that involve enumerating databases in the server.

What happens when database resource limits are reached?

Compute (DTUs and eDTUs / vCores)

When database compute utilization (measured by DTUs and eDTUs, or vCores) becomes high, query latency increases and can even time out. Under these conditions, queries may be queued by the service and are provided resources for execution as resource become free. When encountering high compute utilization, mitigation options include:

- Increasing the performance level of the database or elastic pool to provide the database with more compute resources.
- Optimizing queries to reduce the resource utilization of each query.

Storage

When database space used reaches the max size limit, database inserts and updates that increase the data size fail and clients receive an error message. Database SELECTS and DELETES continue to succeed. When encountering high space utilization, mitigation options include:

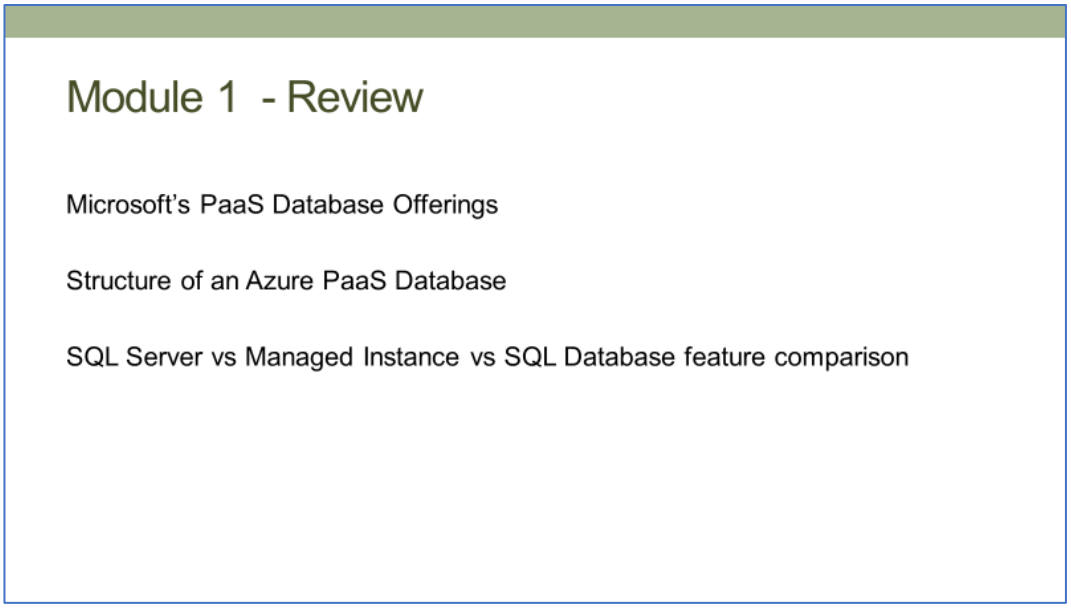
- Increasing the max size of the database or elastic pool, or add more storage.
- If the database is in an elastic pool, then alternatively the database can be moved outside of the pool so that its storage space is not shared with other databases.
- Shrink a database to reclaim unused space.

Sessions and workers (requests)

The maximum number of sessions and workers are determined by the service tier and performance level (DTUs and eDTUs). New requests are rejected when session or worker limits are reached, and clients receive an error message. While the number of connections available can be controlled by the application, the number of concurrent workers is often harder to estimate and control. This is especially true during peak load periods when database resource limits are reached and workers pile up due to longer running queries. When encountering high session or worker utilization, mitigation options include:

- Increasing the service tier or performance level of the database or elastic pool.
- Optimizing queries to reduce the resource utilization of each query if the cause of increased worker utilization is due to contention for compute resources.

These limits are not cast in stone. To obtain more DTU /eDTU quota, vCore quota, or more servers than the default amount, a new support request can be submitted in the Azure portal for the subscription with issue type “Quota”

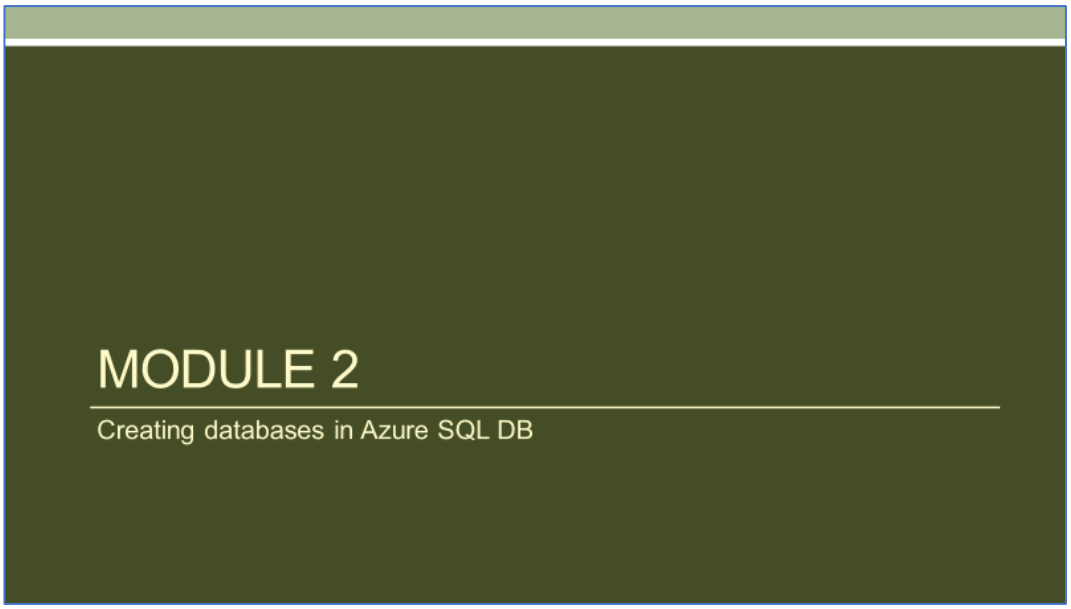


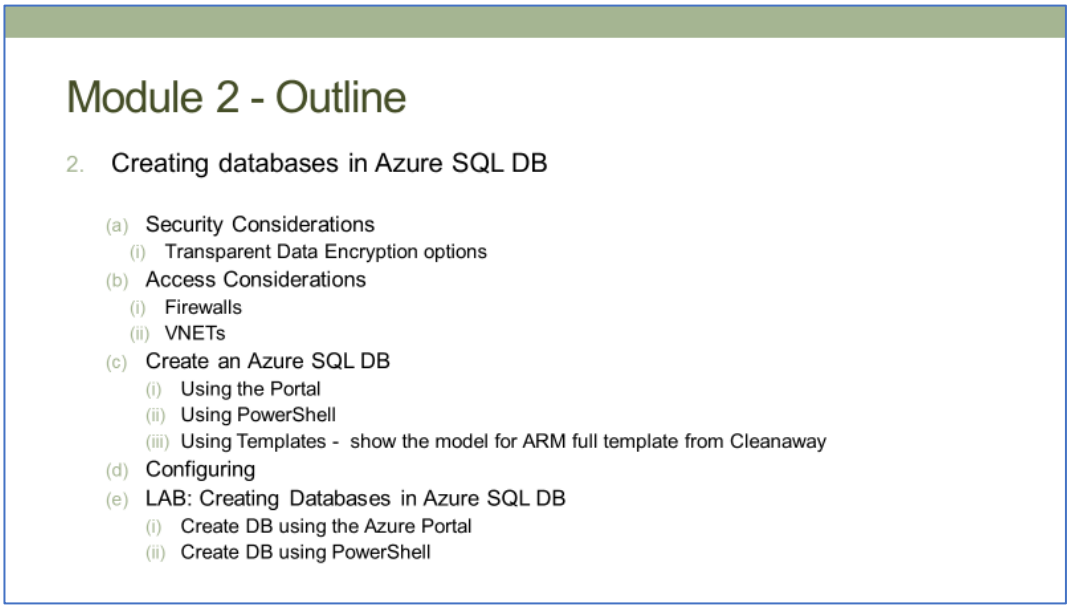
Summary

In this module we explored the types of PaaS database that Microsoft provides in Azure.

These all have similar structures so the themes that we will go on to learn will resonate across all flavours – although each has their own platform specific properties and processes.

We introduced the Azure SQL Database specific concepts that are different to the onpremises SQL Server instances and also how Managed Instance is a closer fit.





Summary

This module will cover how we prepare for and how we then go about creating PaaS databases in Azure.

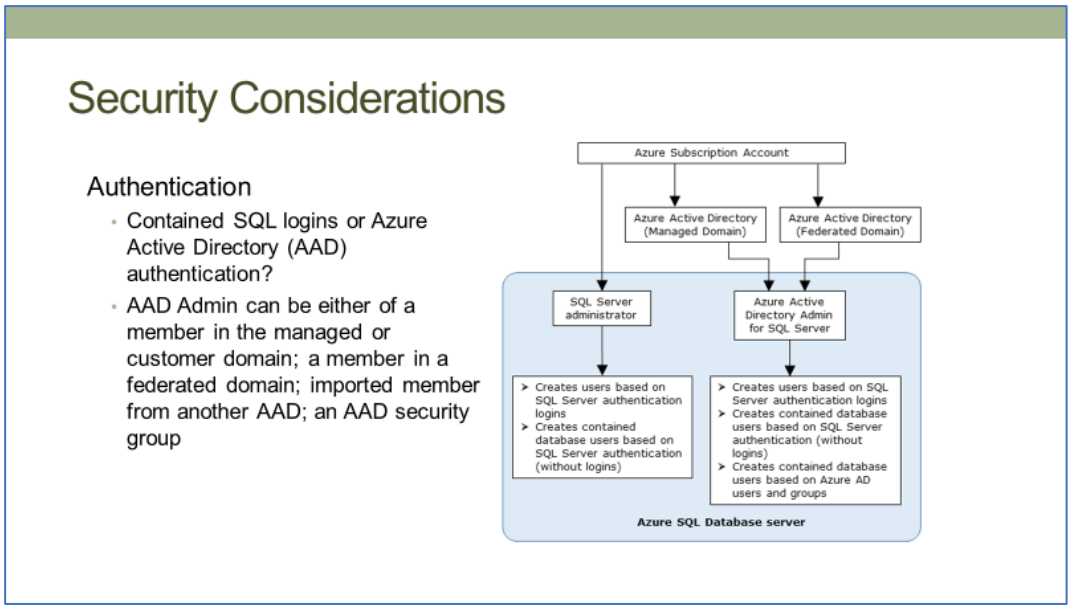
Our planning will need to address how we want to deal with Security and as a result of this what other aspects need to be identified and confirmed – such as how will the Firewalls be configured and where will the client access come from?

We will then look at the different ways that we can create Azure SQL DB databases and how to configure them once they have been created.

There is a LAB that goes with this Module where you can practice creating your own Azure SQL DBs

Resources

Security:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-security-overview
TDE:	https://docs.microsoft.com/en-us/azure/sql-database/transparent-data-encryption-azure-sql
Access Control:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-control-access
Authentication:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-aad-authentication
Firewalls & VNETs	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-firewall-configure
PowerShell References:	https://docs.microsoft.com/enus/powershell/module/azurerm.sql
ARM SQL Templates:	https://docs.microsoft.com/en-us/azure/templates/microsoft.sql/servers
Connection Libraries:	https://docs.microsoft.com/en-us/azure/sqldatabase/sql-database-connect-query



Before we can go ahead and create our Azure SQL DB, we need to know how we plan to access the database and how we therefore secure it.

Authentication:

SQL Authentication is almost identical to the SQL Authentication used by SQL Server. However, for Azure SQL DB the login/password is usually stored in the database (contained login). There is also scope to create the login within the master database of the logical server which means that the same login can be used for any of the databases created within the logical server boundary.

Similarly, to SQL Server, there is also a SQL login that has unrestricted administrator access to the logical server and to each of the databases created within the logical server boundary. This can be named to suit rather than being fixed as "sa" and is defined at creation of the logical server. However, there is not the same mapping of the other Server Rolls that exist in SQL Server – only two roles exist:

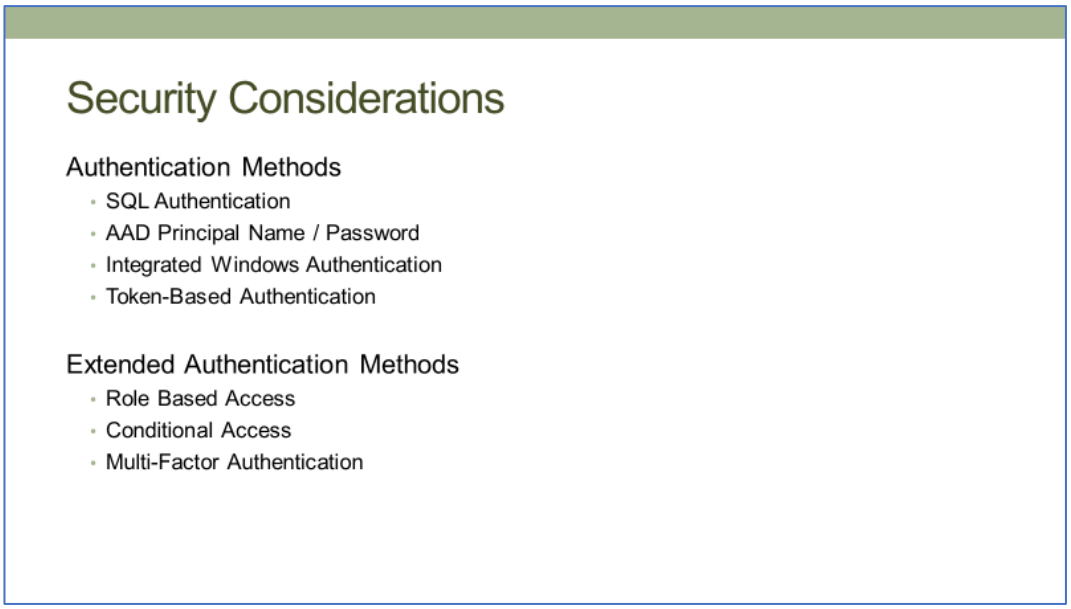
dbmanager	members can create new databases within the logical server boundary
loginmanager	members can create new logins at the logical server level which can then be granted access to each of the databases

Where the tenant has been configured with an AAD service, then Active Directory authentication can also be configured. This requires either an AAD security group or a specific AAD user to be allocated as the Active Directory Admin for the logical server. The users' AAD account can then be created as a contained login in each DB but since the source of authentication is AAD, no logical server level login needs to be created for them to access multiple databases within the logical server boundary.

Best Practice when using AAD authentication is to use a dedicated Azure AD Group as the AAD Admin. To add additional AAD based users to the database, you must be logged in with an AAD user with at least *ALTER ANY USER* permission – regular SQL logins do not have the access to AAD to be able to validate the proposed AAD database user.

Azure AD limitations related to Managed Instance:

- Azure AD server principals is currently in public preview. This will allow an equivalent to Windows Logins in the on-premises version.
- The Azure AD admin used to setup the managed instance cannot be used to create an Azure AD server principal (login) within the managed instance. You must create the first Azure AD server principal (login) using a SQL Server account that is a sysadmin. This is a temporary limitation that will be removed once Azure AD server principals (logins) become GA.



SQL Authentication

The initial admin account can access any database on the logical server as the database owner. Additional SQL logins are either contained to the database they are created in or, if they are created in them master database, can be used as logins and mapped to users in multiple databases.

With AAD Authentication configured, the authentication mechanism is extended beyond the principal/password mechanism (contained SQL authentication) to three methods:

AAD Principal / Password

This works for users from an AAD managed or federated domain. In this format, the user@domain format is implemented for the principal name and their AAD password used to authenticate.

Where you are currently using SQL authentication, this format is probably the simplest way to adopt AAD authentication for these existing applications.

Active Directory Integrated Authentication

This requires the AAD to be federated with your on-premises domain. It also requires the client machines to be domain-joined.

With this configuration, you get the same seamless single sign-on that you would get for Windows Authentication for a SQL Server instance.

Token-Based Authentication

This allows middle tier services to authenticate to Azure SQL DB by obtaining a token from AAD. It also allows for more sophisticated scenarios such as certificate-based authentication.

It requires the following steps to be completed:

- Register the application with AAD to obtain a client ID. This must be in the same directory that the Azure SQL DB is associated with.
- Create a database user representing the application from an AAD identity
- Create a certificate on the client computer that will run the application • Add the certificate as a key for the application

With AAD Authentication in place, access can be further controlled by the use of Conditional Access. This enforces additional policies to the connections such as whether MFA is required, whether the client device must be domain-joined or whether the device is marked as compliant (or a combination of these).

Azure Role Based Access is used to restrict management of the service itself within Azure. RBAC controls are not pushed down to the Azure SQL Database level.

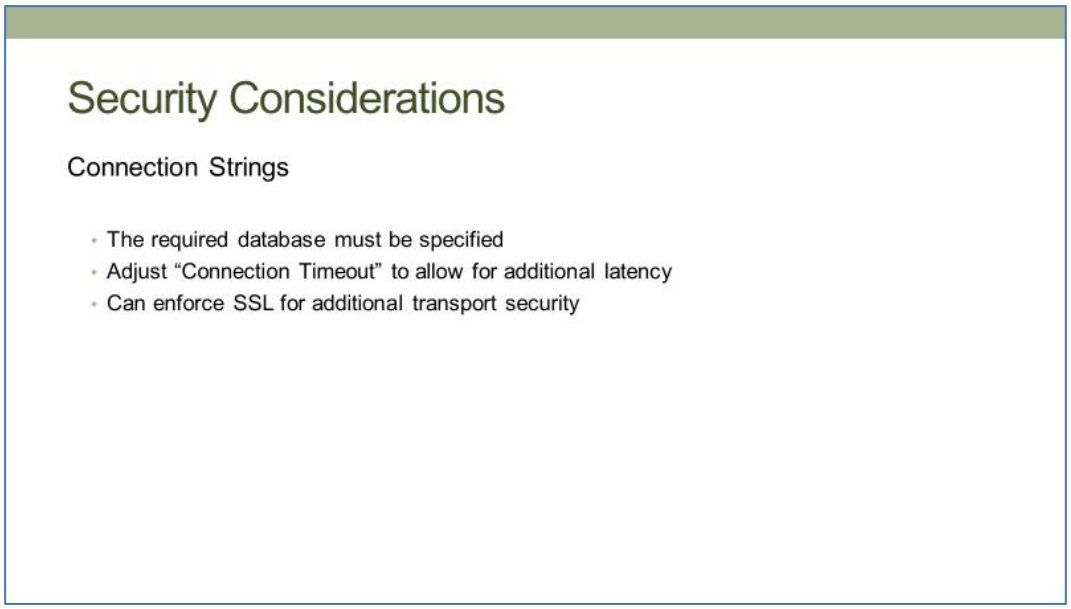
Conditional Access is configured in Azure AD where there can be multiple Conditional Access Policies covering multiple services and/or Groups and Users. This can also be used to block selected Groups/Users from having access to Azure SQL DB. Once the AAD Group/User is selected then the Azure Cloud Apps that the policy relates to is selected – in our case this will be Azure SQL Database. The access controls then define the additional access restrictions/requirements.

Note that Full conditional access requires a Premium Azure Active Directory (Azure

AD). Limited MFA is available with a standard Azure AD. Also, SSMS and

SqlPackage.exe are the only tools currently enabled for MFA through Active Directory Universal Authentication.

Multi-Factor Authentication can also be used to enable Microsoft Accounts that have been added as guests to the AAD to be able to use AAD Authentication when using supported tools to connect. For your own application development, Active Directory Authentication Library (ADAL) version 3.13.9 or later supports Active Directory Universal authentication. Azure AD users for B2B scenarios (including MSA accounts) can only connect to Azure SQL Database as members of an Azure AD Group.



With **Connection Strings**, if you are following best practices and defining contained users within each Azure SQL Database, then you will need to include the database name in the connection string. Some examples of connection strings for the different authentication types are shown below (showing ADO.NET samples):

SQL Authentication:
string ConnectionString =
@"Data Source=myservername.database.windows.net; Authentication=Sql Password; Initial Catalog=testdb; UID=myuser@myservername; PWD=MyPassWord!";

Active Directory Password Authentication:
string ConnectionString =
@"Data Source=myservername.database.windows.net; Authentication=Active Directory Password; Initial Catalog=testdb; UID=myuser@mydomain.com; PWD=MyPassWord!";

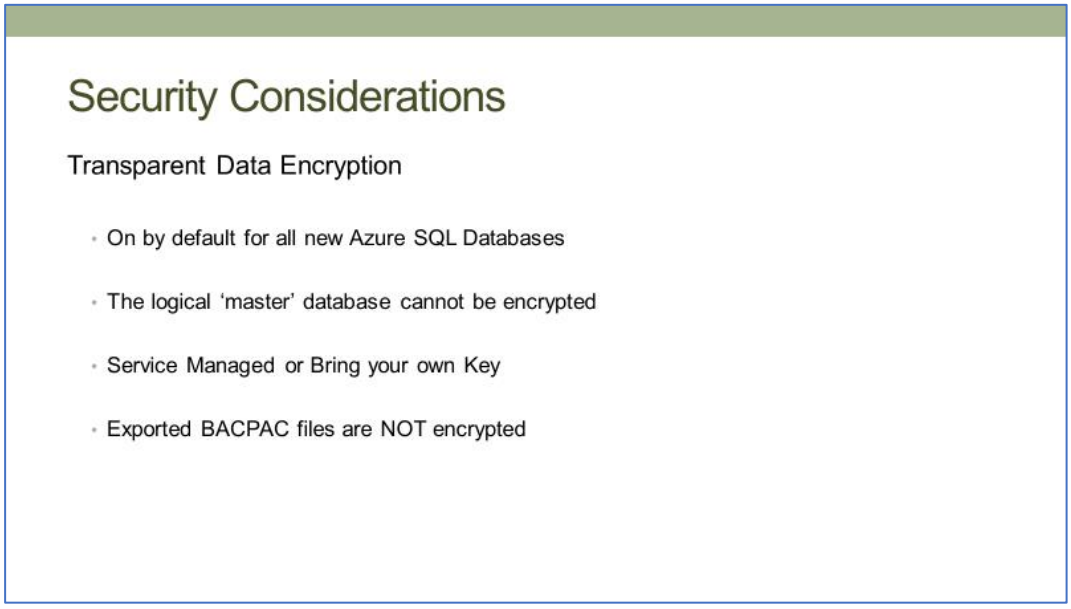
Active Directory Integrated Authentication:
string ConnectionString =
@"Data Source=myservername.database.windows.net; Authentication=Active Directory Integrated; Initial Catalog=testdb; ";

Azure AD Token Authentication:
String accessToken = TokenFactory.GetAccessToken();
string ConnectionString = @"Data Source=myservername.database.windows.net; Initial Catalog=testdb;"
SqlConnection conn = new SqlConnection(ConnectionString);
conn.AccessToken = accessToken conn.Open();

For each of these methods, the "Connection Timeout" property can be added to alter the default value of 15 seconds. There are also optional parameters for "Trust Server Certificate" and "Encrypt" which can be used to enforce the transport security. However, note that Azure SQL Database does not support unencrypted connections no matter what the client settings are.

Azure SQL Database provides a valid certificate to ensure TLS v1.2 is used when connecting with a recent release of each connection library (ADO.Net 4.6 or later, JDBC 4.2 or later, ODBC 11 or later. Node.js 12.7 also supports TLS v1.2, as does PHP 5.6. Support for ODBC on Linux was added in version 13).

To optimise the connection and bypass some of the pre-login handshakes, you should include "Encrypt=true; TrustServerCertificate=false" or equivalent in your connection string.



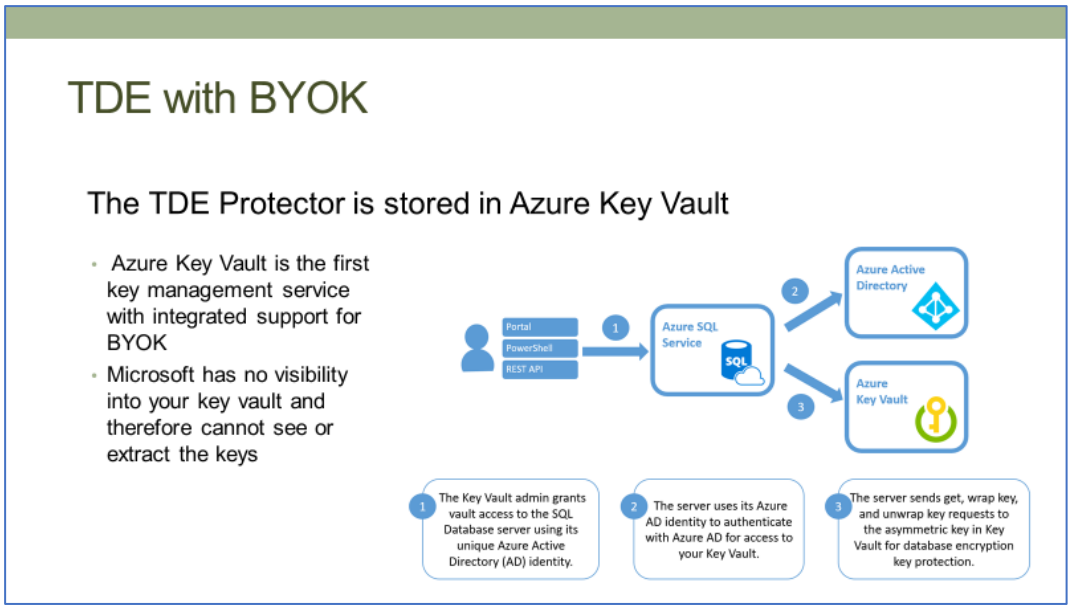
Transparent Data Encryption (TDE) has been a feature of on-premises SQL Server since version 2008. It ensures that data "at-rest" is encrypted and secured – this includes data files and backups. However, it does not encrypt the data within the database which means that a user with permissions can see all of the data within a database with TDE enabled.

The basic configuration for TDE in Azure SQL Database is much simpler than the onpremises version as it is enabled by default for new databases and otherwise can be enabled with a click of a button within the Azure Portal (e.g. after import of a BACPAC).

The default setting for TDE is that the database encryption key is protected by a **Service Managed built-in server certificate**. This is unique to the logical SQL Server, however if an Azure SQL Database is configured for Geo-Replication then both the primary and the secondary databases are protected by the primary database's parent server key. Also, all Azure SQL Databases created in the same logical SQL Server share the same certificate.

For Service Managed TDE, Microsoft automatically manages key rotation at least every 90 days. This includes identifying and moving the keys as needed to support Geo-Replication and restores.

However, you can take control of this and manage your own keys with support for **Bring your own Key (BYOK)**. This allows you to have full control over the keys used and who has access to those keys. Some regulatory compliance certifications may require separation of management of keys from the data that they are used to encrypt and the BYOK configuration for TDE can ensure that this is met.



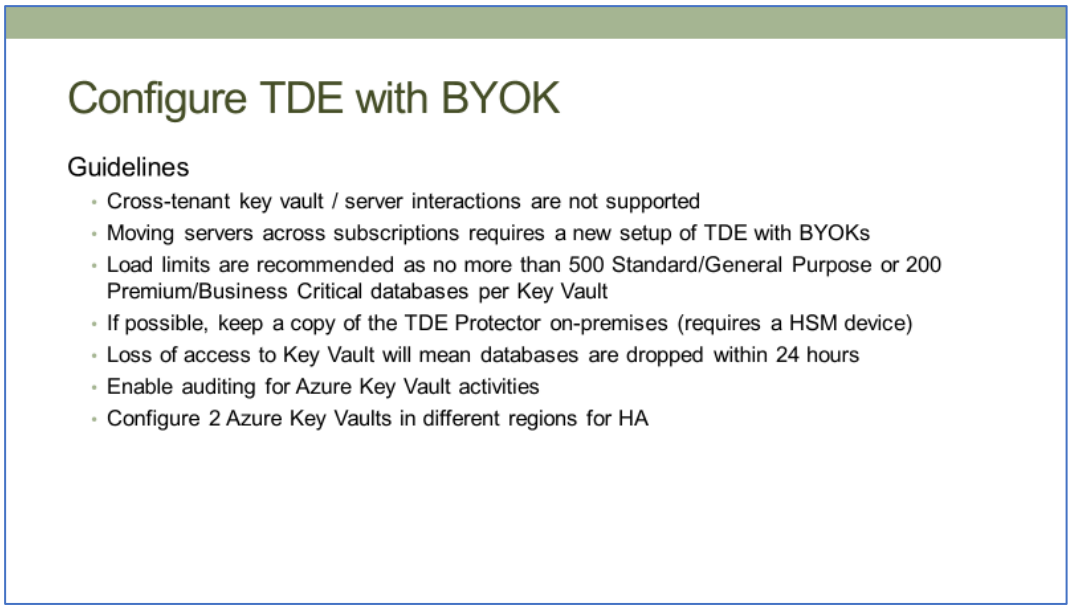
BYOK enables control over the key management tasks:

- Key rotations
- Key vault permissions
- Key deletions
- Auditing / reporting on encryption keys
- Central key management

The TDE Database Encryption Key (DEK) is stored on the boot page of a database and is encrypted and decrypted by the TDE protector. The TDE Protector is stored in Azure Key Vault ***and never leaves the key vault***. If the server's access to the key vault is revoked, a database cannot be decrypted and read into memory. Moving from Service Managed to BYOK is an online operation and does not require re-encryption of the database files as it simply decrypts the DEK using the Service Managed key and then re-encrypts it with the new Azure Key Vault TDE Protector.

When first configuring BYOK, the server sends the DEK of each TDE-enabled Azure

SQL Database to Key Vault with a “wrap-key” request. Key Vault uses the stored TDE Protector to encrypt the DEK and returns the encrypted key which is then stored in the database. Every subsequent request that needs access to the database will send the appropriate Get, Wrap Key or Unwrap request to the Key Vault and receives the appropriate response to allow access to the database. If the database is unable to communicate with Azure Key Vault (either through service issues or revocation of permissions) then all connections to the logical SQL Server are cut-off.



Guidelines for configuring TDE with BYOK

General Guidelines

- Ensure Azure Key Vault and Azure SQL Database are going to be in the same tenant. Cross-tenant key vault and server interactions are not supported.
- Decide which subscriptions are going to be used for the required resources – moving the server across subscriptions later requires a new setup of TDE with BYOKs.
- When configuring TDE with BYOK, it is important to consider the load placed on the key vault by repeated wrap/unwrap operations. For example, since all databases associated with a logical server use the same TDE protector, a failover of that server will trigger as many key operations against the vault as there are databases in the server. Based on our experience and documented key vault service limits, we recommend associating at most 500 Standard / General Purpose or 200 Premium / Business Critical databases with one Azure Key Vault in a single subscription to ensure consistently high availability when accessing the TDE protector in the vault.
- Recommended: Keep a copy of the TDE protector on premises. This requires an HSM device to create a TDE Protector locally and a key escrow system to store a local copy of the TDE Protector.

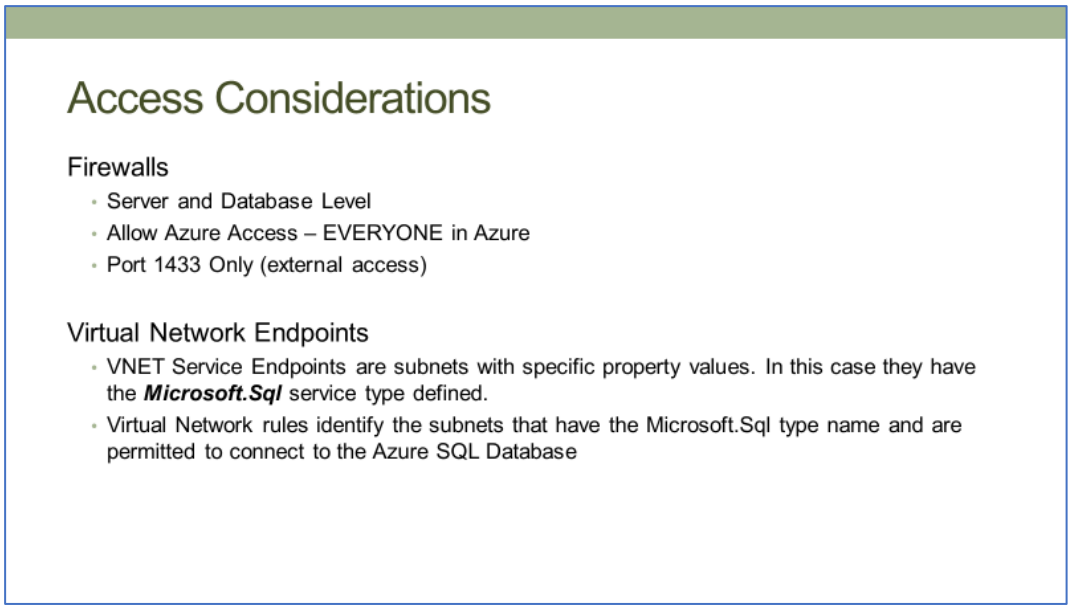
Guidelines for configuring Azure Key Vault

- Create a key vault with soft-delete enabled to protect from data loss in case of accidental key – or key vault – deletion. You must use PowerShell to enable the “soft-delete” property on the key vault (this option is not available from the AKV Portal yet – but required by SQL):
- Soft deleted resources are retained for a set period of time, 90 days unless they are recovered or purged.
- The recover and purge actions have their own permissions associated in a key vault access policy.
- Set a resource lock on the key vault to control who can delete this critical resource and help to prevent accidental or unauthorized deletion.
- Grant the logical server access to the key vault using its Azure Active Directory (Azure AD) Identity. When using the Portal UI, the Azure AD identity gets automatically created and the key vault access permissions are granted to the server. Using PowerShell to configure TDE with BYOK, the Azure AD identity must be created and completion should be verified.
- If the Azure AD Identity is accidentally deleted or the server’s permissions are revoked using the key vault’s access policy, the server loses access to the key vault, and TDE encrypted databases are dropped within 24 hours.

- Configure Azure Key Vault without a VNET or firewall.
- Enable auditing and reporting on all encryption keys: Key Vault provides logs that are easy to inject into other security information and event management (SIEM) tools. Operations Management Suite (OMS) Log Analytics is one example of a service that is already integrated.
- To ensure high-availability of encrypted databases, configure each logical server with two Azure Key Vaults that reside in different regions.

Guidelines for configuring the TDE Protector (asymmetric key) stored in Azure Key Vault

- Create your encryption key locally on a local HSM device. Ensure this is an asymmetric, RSA 2048 key so it is storable in Azure Key Vault.
- Escrow the key in a key escrow system.
- Import the encryption key file (.pfx, .byok, or .backup) to Azure Key Vault.
- For testing purposes, it is possible to create a key with Azure Key Vault, however this key cannot be escrowed, because the private key can never leave the key vault. Always back up and escrow keys used to encrypt production data, as the loss of the key (accidental deletion in key vault, expiration etc.) results in permanent data loss.
- Use a key without an expiration date – and never set an expiration date on a key already in use: once the key expires, the encrypted databases lose access to their TDE Protector and are dropped within 24 hours.
- Ensure the key is enabled and has permissions to perform get, wrap key, and unwrap key operations.
- Create an Azure Key Vault key backup before using the key in Azure Key Vault for the first time.
- Create a new backup whenever any changes are made to the key (for example, add ACLs, add tags, add key attributes).
- Keep previous versions of the key in the key vault when rotating keys, so older database backups can be restored. When the TDE Protector is changed for a database, old backups of the database are not updated to use the latest TDE Protector. Each backup needs the TDE Protector it was created with at restore time.
- Keep all previously used keys in Azure Key Vault after changing back to service-managed keys. This ensures database backups can be restored with the TDE protectors stored in Azure Key Vault. TDE protectors created with Azure Key Vault have to be maintained until all stored backups have been created with service-managed keys.
- Make recoverable backup copies of these keys using BackupAzureKeyVaultKey.
- To remove a potentially compromised key during a security incident without the risk of data loss, follow the steps at https://docs.microsoft.com/en-us/azure/sql-database/transparent-dataencryption-byok-azure-sql-remove-tde-protector.



The default status for Azure SQL Database is to be locked down at creation. Any access to the logical SQL Server or the individual databases must be explicitly granted – initially through the use of Firewalls.

Firewalls exist at two levels – the Server Level (the rules are stored in the master database) and optionally at the Database Level (stored in the individual databases, which may include the master database). There has to be at least one Server Level firewall rule created to allow access. No matter where your client is located – on premises, in Azure or another cloud – it has to pass through the internet –facing firewall to gain access to the required Azure SQL Database.

To permit your applications that are hosted in Azure to connect to your Azure SQL Database, you need to enable the option for Azure Connections – however it is important to remember when you do this that this means access to the WHOLE of Azure – not just your tenant or subscription so you must combine this with appropriate permissions and security for the logins and users – ideally using Azure AD to limit the scope of the user accounts.

Also note that Azure SQL Database only listens on Port 1433. This does mean that the firewall rules only have to define IP Address ranges and not worry about Ports as well.

Note however that this is for clients originating OUTSIDE the Azure Cloud Boundary. For clients INSIDE the Azure Cloud Boundary, it uses ***Direct Route*** connections to interact with Azure SQL Database. The process is that the client (ADO.NET 4.5 or later) initiates an interaction with the Azure Cloud and receives a dynamically assigned port number (in the ranges 11000-11999 or 14000-14999). The client then establishes the connection with Azure SQL Database on that Port Number with no middleware in between. Therefore, it is important to ensure that your client VM (for example) allows outbound access on the port ranges above.

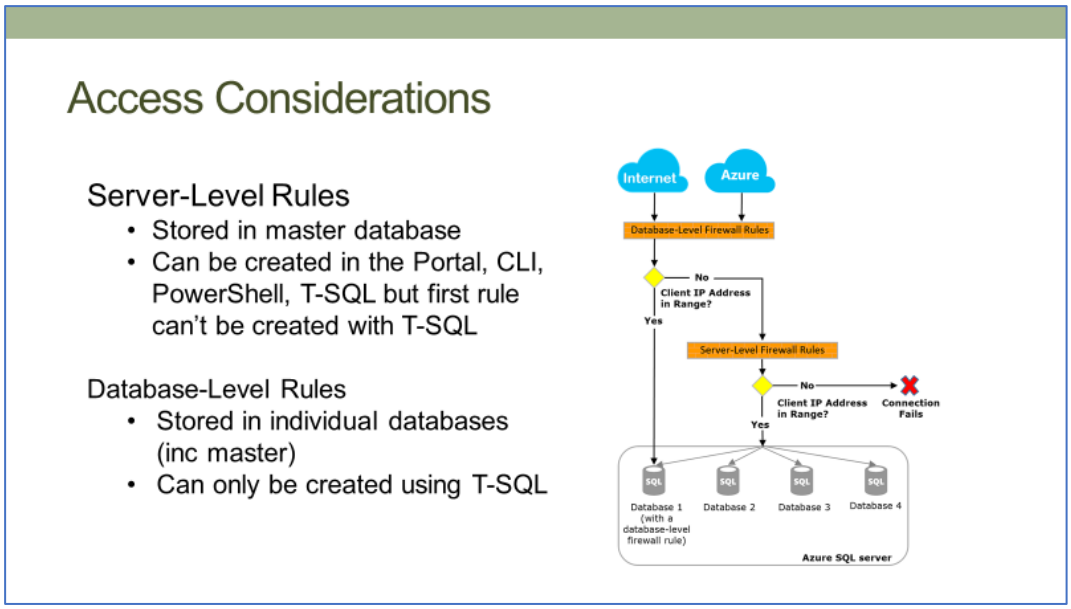
In addition to IP Address rules, the firewall can also manage ***virtual network rules***. These are based on Virtual Network service endpoints and in some cases may be preferable to the IP Address rules.

By using a VNET Service Endpoint, the access from ALL Azure services can be made more granular and restricted to only the defined Subnets. Where you have other services within the Azure private network that need to connect to your Azure SQL Database and you don't want to

enable Azure Connections then you need to specify the PUBLIC IP Address for that service in your Firewall Rules. However, most services will have dynamic external IP Addresses which means that you could be redefining Firewall Rules every time e.g. a VM is restarted. Using the VNET Service Endpoint allows the private IP Address within the defined subnet to be granted access and no need to manage the dynamic IP Address ranges.

There are a number of restrictions to using a VNET Service Endpoint though:

- The VNET Rules are applied at the Server Level and not the Database Level – this may open up more access than is desired.
- Azure Web Apps, despite being able to be mapped to a private IP Address within a VNET/Subnet will have a Public IP Address when they try to connect to Azure SQL Database – therefore for Web Apps the "Allow Azure Services to access server" must be enabled.
- Although each VNET Rule defined in Azure SQL Database can reference a different Subnet, all the referenced Subnets must be hosted in the same Azure Region as the Azure SQL Database.
- If you also have Azure Database for MySQL or Azure Database for PostgreSQL instances then enabling VNET Service Endpoints for Azure SQL Database will also enable them for the other PaaS database services. This will result in connection failures to MySQL and PostgreSQL. You must then configure virtual network rules for MySQL/PostgreSQL to reenable the connections.
- VNET Rules do not currently apply to Site-to-Site VPNs or Express Route connections – IP Address based rules must be used for these.
- The Network Security Groups (NSG) associated with the VNETs need to permit outbound connections to the Azure SQL Database public IP Addresses. This is most easily done with NSG Service Tags.



The firewall process is as follows:

- The firewall first checks the originating IP address of the request against the database-level firewall rules, for the database that the connection is requesting:
  - If the IP address of the request is within one of the ranges specified in the database-level firewall rules, the connection is granted to the SQL Database that contains the rule.
  - If the IP address of the request is not within one of the ranges specified in the database-level firewall rule, the server-level firewall rules are checked. If the IP address of the request is within one of the ranges specified in the server-level firewall rules, the connection is granted. Server-level firewall rules apply to all SQL databases on the Azure SQL server.
  - If the IP address of the request is not within the ranges specified in any of the database-level or server-level firewall rules, the connection request fails.

It is possible to grant a client IP Address access to only a single Azure SQL Database on the server. To achieve this, you should define the Database-Level Firewall Rule for the desired database to be beyond the range for the Server-Level Firewall Rules while still ensuring that the IP Address of the client machine will remain within the Database-Level range.

When Allow Access from Azure Services is enabled, a Server-Level Firewall Rule with a start and end address equal to 0.0.0.0 will be created. This can be queried within Azure SQL Database to confirm whether this has been enabled.

When an application from Azure attempts to connect to your database server, the firewall verifies that Azure connections are allowed by confirming a the rule for 0.0.0.0 exists. If the connection attempt is not allowed, the request does not reach the Azure SQL Database server.

The first Rule needs to be created at an Azure level - either through the Portal, CLI, PowerShell or the REST API. All subsequent rules can then be added using T-SQL.

If your network is connected to the Azure network through use of ExpressRoute, each circuit is configured with two public IP addresses at the Microsoft Edge. The two IP addresses are used to connect to Microsoft Services, such as to Azure Storage, by using Azure Public Peering. To allow communication from your circuit to Azure SQL Database, you must create IP network rules for

the public IP addresses of your circuits. In order to find the public IP addresses of your ExpressRoute circuit, open a support ticket with ExpressRoute by using the Azure portal.

The following points will help you identify whether to use Server-level firewall rules or database-level firewall rules:

Q. Should users of one database be fully isolated from another database?

If yes, grant access using database-level firewall rules. This avoids using server-level firewall rules, which permit access through the firewall to all databases, reducing the depth of your defences.

Q. Do users at the IP address's need access to all databases?

Use server-level firewall rules to reduce the number of times you must configure firewall rules.

Q. Does the person or team configuring the firewall rules only have access through the Azure portal, PowerShell, or the REST API?

You must use server-level firewall rules. Database-level firewall rules can only be configured using Transact-SQL.

Q. Is the person or team configuring the firewall rules prohibited from having highlevel permission at the database level?

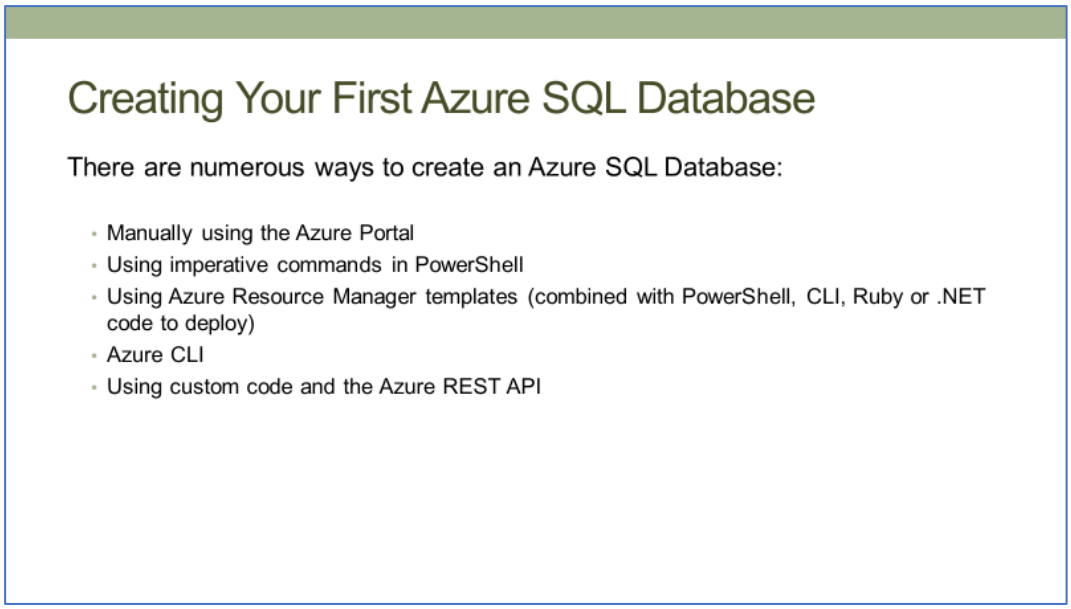
Use server-level firewall rules. Configuring database-level firewall rules using Transact-SQL, requires at least CONTROL DATABASE permission at the database level.

Q. Is the person or team configuring or auditing the firewall rules, centrally managing firewall rules for many (perhaps 100s) of databases?

This selection depends upon your needs and environment. Server-level firewall rules might be easier to configure, but scripting can configure rules at the database-level. And even if you use server-level firewall rules, you might need to audit the databasefirewall rules, to see if users with CONTROL permission on the database have created database-level firewall rules.

Q. Can I use a mix of both server-level and database-level firewall rules?

Yes. Some users, such as administrators might need server-level firewall rules. Other users, such as users of a database application, might need database-level firewall rules.



Azure SQL Databases (and almost all other Azure resources) can be created using a number of techniques.

Possibly the most straight forward and simplest is to use the Azure Portal (https://portal.azure.com). Within the Portal, the blade for creation of a SQL database will prompt you for, and verify each of the items that are needed to create your Azure SQL Database.

Using the Portal, you will be prompted for:

- The database name.
- The subscription in which it will be created.
- Which Resource Group to create the database in. You can select an existing Resource Group or follow the prompts to create a new one.
- Identify whether you need a blank database, a sample copy of AdventureWorks LT or restore from another Azure SQL Database's backup. You cannot restore from an on-premises SQL Server database backup.
- If a logical SQL Server has already been created then this can be selected. Otherwise the option to also create a SQL Server can be selected. This will additionally prompt for a Server Name, the Name and Password for the initial administrator login (this is a SQL login – the AAD Administrator can only be added after creation), the Azure Region for the SQL Server, whether to Allow Azure Services and whether to enable the optional Advanced Threat Detection at this time. The Threat Detection can easily be enabled afterwards if required.
- Whether to use an Elastic Pool. This can be an existing pool to add the new Azure SQL Database to or you can create a new pool by specifying the name and the Performance Tier settings. If an Elastic Pool is not selected then the Performance Tier settings for the single database are presented.
- The final setting that can be configured at this stage is the database collation. The default is "SQL_Latin1_General_CP1_CI_AS"

Using PowerShell imperative code, you would use code similar to the following:

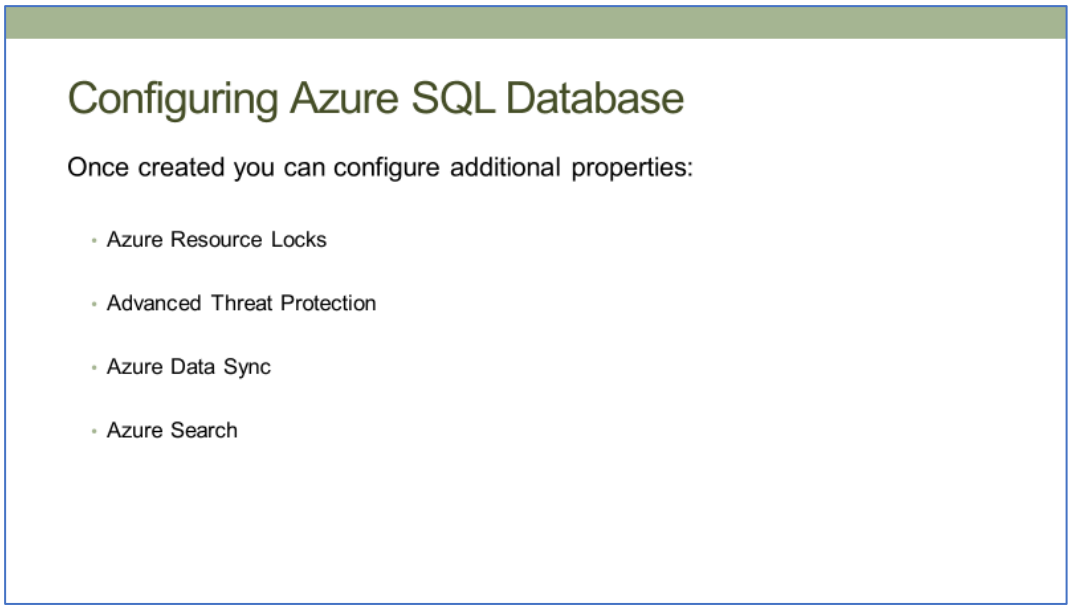
```
New-AzSqlServer          -ServerName <String>
                         -SqlAdministratorCredentials <PSCredential>
                         -Location <String>
New-AzSqlDatabase        -DatabaseName <String>
                         -ServerName <String>
                         -ResourceGroupName <String>
                         -Edition <String>
                         -MaxSizeBytes <Int64>
                         [-VCore <Int32>]
                         [-ComputeGeneration <String>]
                         [-LicenseType <String>]
                         [-CollationName <String>]
```

However, the most complete and thorough method is by using ARM Templates. When using ARM Templates, the template must include the Resource Type "databases“. Optionally you can also define properties such as whether Access from Azure Services is enabled and whether Advanced Threat Detection is switched on by default.

Additional components such as Auditing and Metrics capture can also be included in the remplate.

```
"resources": [
{
        "apiVersion": "2017-10-01-preview",
        "dependsOn": [
                "[concat('Microsoft.Sql/servers/', parameters('serverName'))]“
        ],
        "location": "[parameters('location')]",
        "name": "[parameters('databaseName')]",
        "properties": {
                  "collation": "[parameters('collation')]",
                  "maxSizeBytes": "[parameters('maxSizeBytes')]",
                  "sampleName": "[parameters('sampleName')]",
                  "zoneRedundant": "[parameters('zoneRedundant')]",
                  "licenseType": "[parameters('licenseType’)]”
        },
        "sku": {
                "name": "[parameters('skuName')]",
                "tier": "[parameters('tier’)]”
        },
        "type": "databases“
}
```

DEMO 1: Create an Azure SQL Database using the imperative commands
DEMO 2: Create an Azure SQL Database using ARM templates



Either in the ARM Template or after creation of the SQL Server and the Azure SQL Database, more properties are exposed that can be configured as required. These exist at both the SQL Server and the Azure SQL Database level. Many of these we will cover in later modules – the others are discussed below.

The core tasks post-creation should focus on properly securing your Azure SQL Database.

At the ***SQL Server level***, these cover:

- Failover Groups (Module 5)
- Backup retention Policies (Module 5)
- Active Directory Administrator
- Firewalls and VNETs
- Auditing (Module 7)
- TDE
- Automatic Tuning (Module 8)
- Advanced Threat Protection
- Locks

At the ***Azure SQL Database level***, these cover:

- Geo-Replication (Module 5)
- Azure Data Sync
- Azure Search
- Dynamic Data Masking Policies
- Auditing (Module 7)
- TDE
- Automatic Tuning (Module 8)
- Advanced Threat Protection
- Alerts (Module 7)
- Diagnostics (Module 8)
- Locks

Resource Locks:

Azure resources can have Locks applied to further restrict the actions that can take place. There are two types of Locks

- **Delete** Authorised users can't delete but can read / modify a resource.
- **ReadOnly** Authorised users can read a resource but they cannot edit or delete it.

To enable this functionality (e.g. to prevent accidental deletion of a SQL Server) you need to be either **Owner** or **User Access Administrator**

Advanced Threat Protection:

The suite of Advanced Threat Protection routines have been developed using advanced analytics to be able to identify and classify your data and database activities to provide further insights and notification of suspicious activity.

When enabled at the SQL Server level, Threat Detection is enabled for all Azure SQL Databases related to that SQL Server. Notification details can be added and a growing list of Threat Detection Types can be selected – these cover SQL Injection vulnerabilities, anomalous activities such as logins from an unknown source or brute force attacks, and attempt to exfiltrate data.

At the Azure SQL Database level Data Discovery and Classification is added which will scan the schema of your database and suggest possibly sensitive columns that you should consider for further protection using e.g. Data Masking or Always Encrypted. It is recommended to enable Threat Protection at the SQL Server level unless specific variations in the detection for different Azure SQL Databases is required.

To get the full power of Advanced Threat Detection, it should be combined with Auditing and also have a Storage Account linked to store the results of the Vulnerability Assessment scans. The scans employ a knowledge base of rules that flag security vulnerabilities and highlight deviations from best practices, such as misconfigurations, excessive permissions, and unprotected sensitive data. The rules are based on Microsoft's best practices. These rules also represent many of the requirements from various regulatory bodies to meet their compliance standards. Results of the scan include actionable steps to resolve each issue and provide customized remediation scripts where applicable. An assessment report can be customized for your environment by setting an acceptable baseline for permission configurations, feature configurations, and database settings.

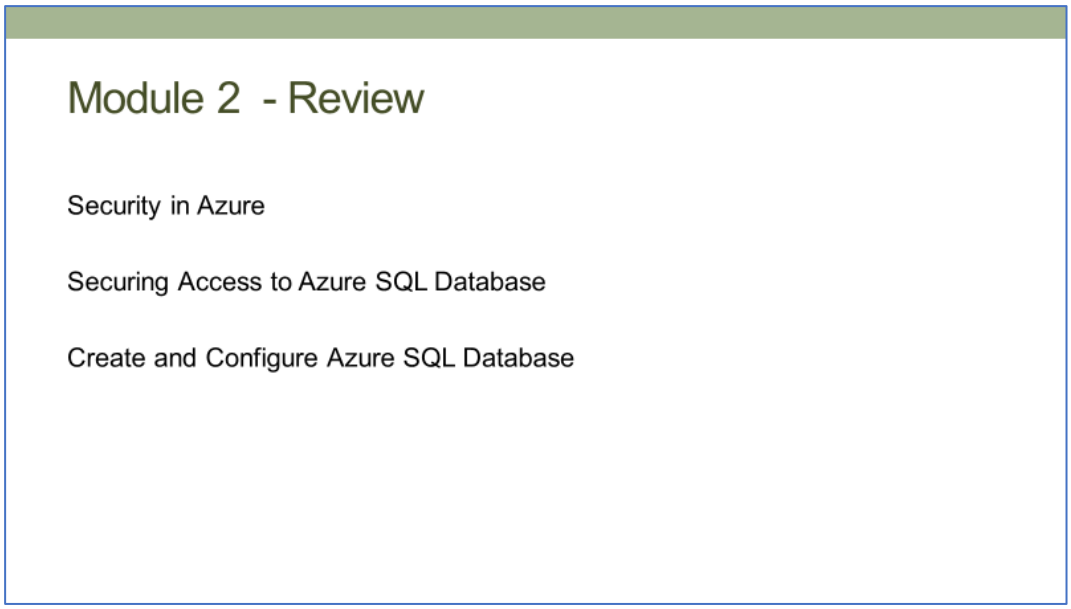
Azure Data Sync:

Azure Data Sync is analogous to a Merge Replication configuration in SQL Server and allows for the single or bi-directional replication of data between an Azure SQL Database and other Azure SQL Databases or a SQL Server database.

Azure Search:

Azure SQL Database is one of the supported Indexers that will crawl and extract searchable data and metadata to populate the Index. Azure Search can also hook into Cognitive Services to use Search for faceted navigation into the data source to simplify the discovery of data.

Azure Data Sync and Azure Search are not covered in this course.



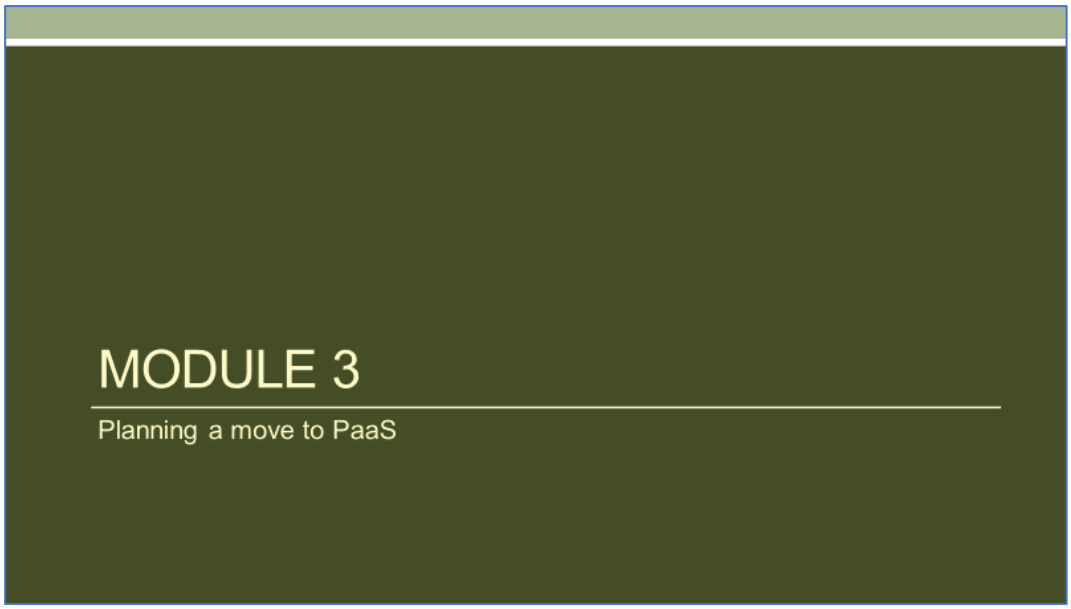
Summary

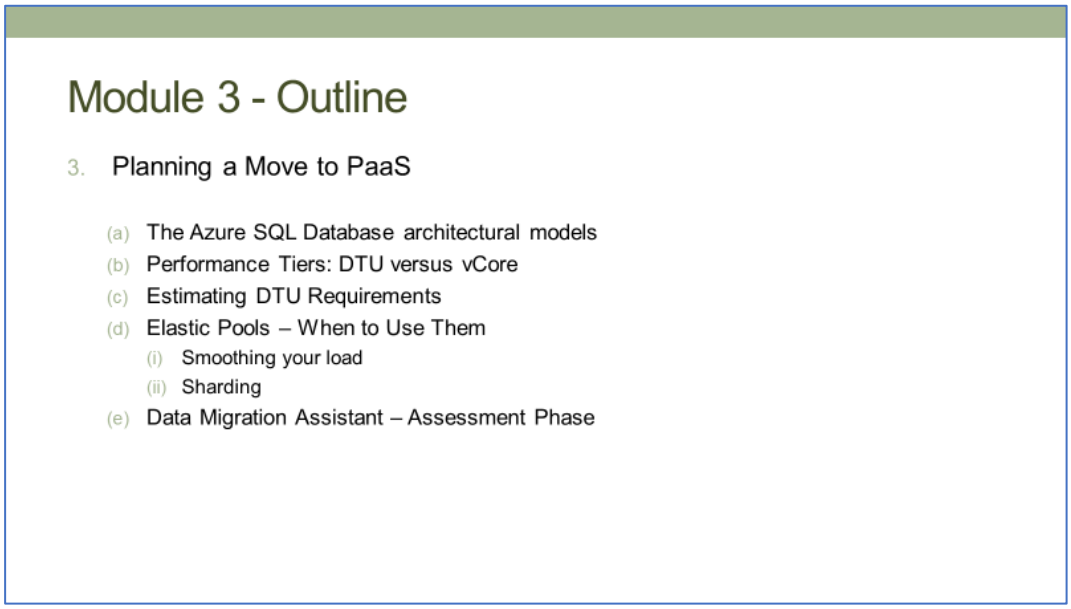
In this module we looked at security related to Azure SQL Database.

There is security in depth though a number of layers and it needs to be carefully planned out.

Azure provides security through its own RBAC controls for each service and within the service we are able to lock it down further with Azure AD authentication, firewalls, VNETs and the internal database provisions of TDE, Always Encrypted, Row-Level Security, Data Masking etc.

Once we understand our security needs we can then start to create our Azure SQL Databases using scripts and/or templates for repeatability.





Introduction

This module will cover how we plan the type of PaaS solution that is appropriate for the workload and the scale of that solution.

There are four architectural models used for Azure SQL Database:

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

This course focusses on the first two options, however we will see the differences that Hyperscale and Serverless can offer.

When considering Azure SQL Database, we need to decide between the predictable scalability of the DTU model against the practical ability of scaling storage and compute independently in the vCore model. For the other PaaS options, we only have the vCore model to choose from. The vCore model obviously translates easily from on-premises or VM-based SQL Servers but we

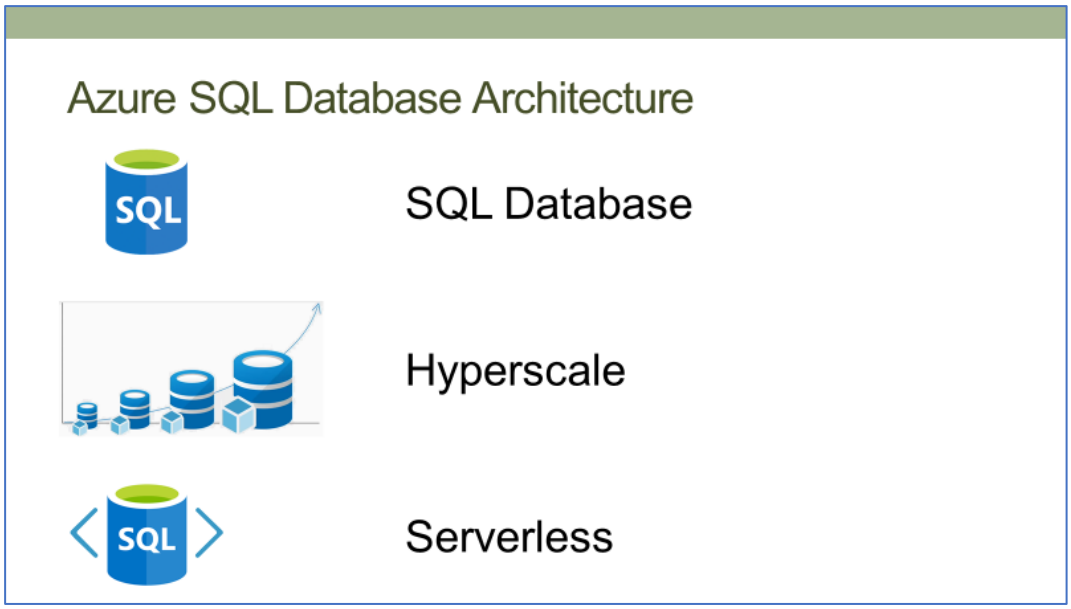
need to make use of other tools to help estimate the DTU requirements if we choose that model.

Elastic Pools are not always a viable option for a group of databases. Depending upon the workload or the required architecture we might need them so we will consider the benefits and the patterns that pick these out.

Finally, we'll look at recent tooling improvements that will help plan the migration from SQL Server to Azure SQL Database.

Resources

Azure SQL Database Hyperscale:	https://docs.microsoft.com/enus/azure/sql-database/sql-database-service-tier-hyperscale
Azure SQL Database Serverless:	https://docs.microsoft.com/enus/azure/sql-database/sql-database-serverless
vCores & DTUs:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers
Server Resource Limits:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-resource-limits-logical-server
Scale Resource Tiers:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-scale-resources
Reserved Capacity:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-reserved-capacity
DTU Calculator:	https://dtucalculator.azurewebsites.net/
Data Migration Assistant:	https://docs.microsoft.com/enus/sql/dma/dma-overview?view=sql-server-2017
Microsoft Data Migration Blog:	https://techcommunity.microsoft.com/t5/Microsoft-DataMigration/bg-p/MicrosoftDataMigration
Elastic Pools:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-pool
SaaS Design Patterns:	https://docs.microsoft.com/en-us/azure/sql-database/saas-tenancy-app-design-patterns
Sharding:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-scale-introduction



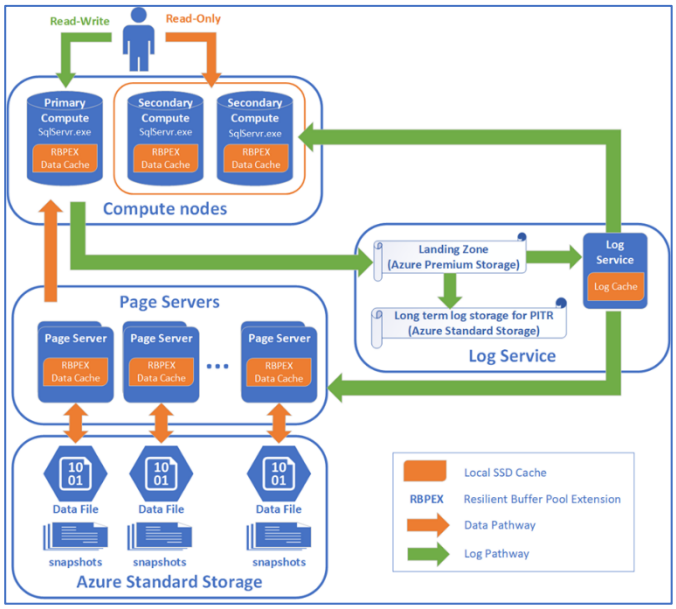
Azure SQL Database:

The core offering comes in a number of flavours as well. IN the DTU model there is

Basic/Standard and Premium. These match with General Purpose and Business Critical in the vCore model. Each of these has a slightly different architecture behind it as we will see in Module 5. The primary difference is the way that compute and storage is put together which leads to different performance profiles.

Hyperscale:

The Hyperscale Architecture is shown below:



The separation of storage, compute and logging results in a very fast and massively scalable database solution. The Hyperscale service tier removes many of the practical limits traditionally seen in cloud databases. Where most other databases are limited by the resources available in a single node, databases in the Hyperscale service tier have no such limits. With its flexible storage architecture, storage grows as needed. In fact, Hyperscale databases aren't created with a defined max size. A Hyperscale database grows as needed - and you are billed only for

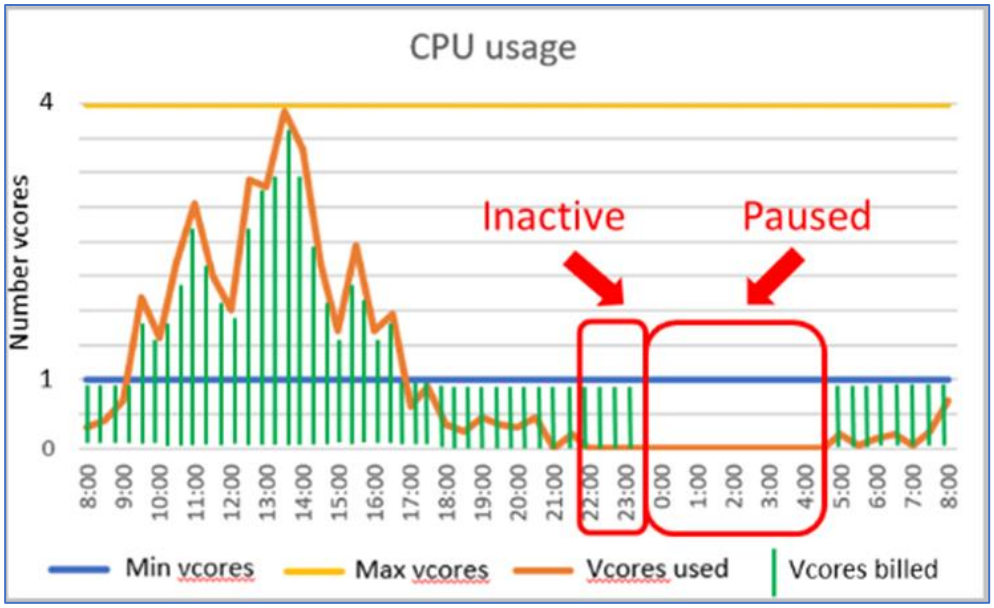
the capacity you use. For read intensive workloads, the Hyperscale service tier provides rapid scale-out by provisioning additional read replicas as needed for offloading read workloads.

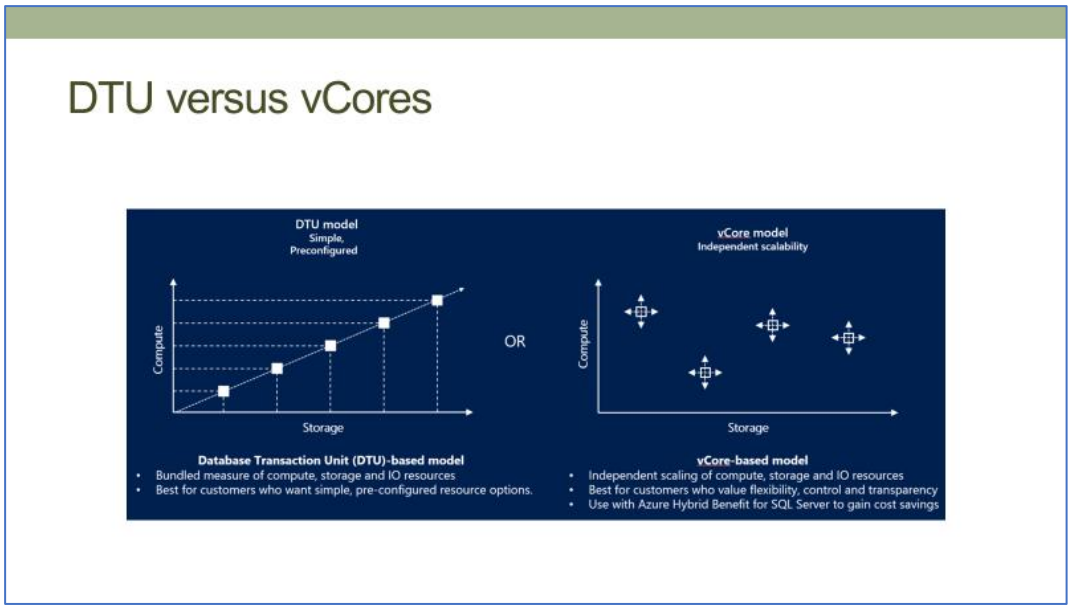
The separation of each layer also allows for a different approach to backups as the data tier can be snapshotted for near instantaneous backups and restores. The Log Service also removes the blockage on throughput through the direct updates to the Page Servers.

Serverless:

The newest member of the Azure SQL Database family is Serverless. As the name suggests, this operates more like an on-demand database engine. It is suited to workloads with intermittent, unpredictable usage patterns that can afford some delay in compute warm-up after idle usage periods.

An additional scenario where Serverless could be used is for new single databases without usage history. If the compute sizing is difficult or not possible to estimate prior to deployment, then it could be deployed to Deployed initially to Serverless until a profile is obtained of the usage and the required resources.





The primary difference between the DTU and vCore models is that in the DTU model the performance tier is a blend of CPU, Memory and IOPS performance whereas the vCore model allows storage and IOPS to be scaled independently from the compute (CPU and Memory) resources.

Although the DTU model doesn't explicitly define how much CPU power or how much memory is available at each tier, the scalability is linear – e.g. increasing the Standard Tier from 50 DTUs to 100 DTUs will double the performance available to the database. The SQL license cost is included within the price point for the selected performance tier but is not explicitly called out. In the vCore model, the license cost is directly linked to the number of vCores and as a result it opens up the option to use Azure Hybrid Benefits and bring existing SQL Server core licenses to Azure SQL Database to help save costs.

As a rule of thumb, every 100 DTU in the Standard tier equates roughly to 1 vCore in the General Purpose tier, and every 125 DTU in the Premium tier equates roughly to 1 vCore in the Business Critical tier. Where your DTU requirements are less than 1 vCore, it is not recommended to use the vCore model.

DTU

Advantages: Simple to manage and scale. Built in License costs. Linear scalability.
Disadvantages: All resources scale together – need more storage then you get more compute.

vCores

Advantages: Azure Hybrid Benefits. Scale storage and compute independently.
Disadvantages: Potentially more expensive (do you have SA?). Backup Storage priced separately. Not available in all Azure Regions at this time.

DTU Tiers

	Basic	Standard	Premium
Uptime SLA	99.99%	99.99%	99.995%
Backup retention	7 - 35 days	7 - 35 days	7 - 35 days
DTU Range	5	10 to 3000	125 - 4000
CPU	Low	Low, Medium, High	Medium, High
IO throughput (approximate)	2.5 IOPS per DTU	2.5 IOPS per DTU	48 IOPS per DTU
IO latency (approximate)	5 ms (read), 10 ms (write)	5 ms (read), 10 ms (write)	2 ms (read/write)
Columnstore indexing	N/A	S3 and above	Supported
In-memory OLTP	N/A	N/A	Supported
Maximum Storage Size	2 GB	1 TB	4 TB

The core performance characteristics of the different Tiers in the DTU model are shown in the table. Note that within each Performance Tier, the level is selectable in distinct points within the documented range.

If using an Elastic Pool, there are sub-limits within the Pool's eDTU overall limit.

There is some documentation around the reference architecture and performance characteristics that are used to determine the DTU values at https://docs.microsoft.com/en-us/azure/sql-database/sql-database-service-tiers-dtu.

vCore Tiers

	General Purpose	Business Critical
Uptime SLA	99.99%	99.995%
Compute Options	Gen 4: 1 to 24 vCores Gen 5: 1 to 80 vCores	Gen 4: 1 to 24 vCores Gen 5: 1 to 80 vCores
Memory	Gen 4: 7GB per vCore Gen 5: 5.5GB per vCore	Gen 4: 7GB per vCore Gen 5: 5.5GB per vCore
Storage	Premium Remote Storage: 5GB – 4TB	Local SSD Storage: 5GB – 4TB
IO throughput (approximate)	500 IOPS per vCore	5000 IOPS per vCore
Max IOPS (approximate)	7000 IOPS	200000 IOPS
IO latency (approximate)	5-10 ms	1-2 ms
Columnstore indexing	Supported	Supported
In-memory OLTP	N/A	Supported
Backups	7 – 35 days configurable	7 – 35 days configurable
Maximum Storage Size	1 TB	4 TB

vCores are provided with two different physical architectures:

Gen 4

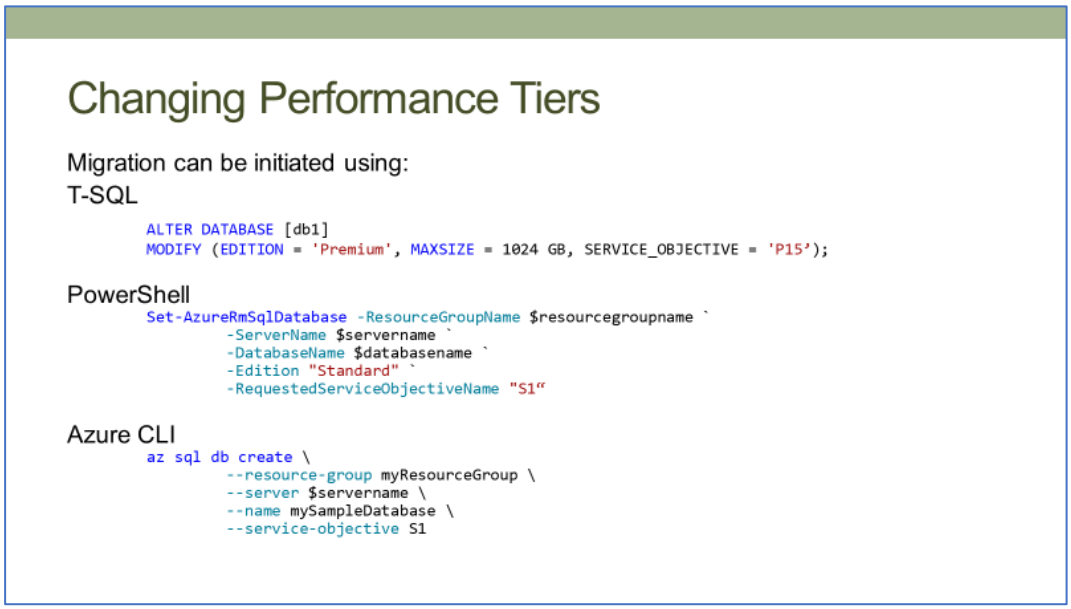
These are based on the Intel E5-2673 v3 (Haswell) 2.4 GHz processors. Each vCore maps to a single physical core in these processors and each vCore has 7GB of memory and uses attached SSDs for storage.

Gen 5

These are based on the Intel E5-2673 v4 (Broadwell) 2.3 GHz processors. Each vCore maps to a hyperthread logical processor and each vCore has 5.5GB of memory and uses fast eNVM SSDs for storage.

The Storage Sizing for vCore based Azure SQL Databases has some specific considerations that you need to be aware of:

- When you configure the required database size an additional 30% is automatically added for the Transaction Log file
- In the General Purpose Tier, tempdb uses an attached SSD. Storage cost is included in the cost and is not applied separately
- In the Business Critical Tier, tempdb shares the attached SSD. Storage cost is included in the cost and is not applied separately
- Point-in-Time restores are supported within the configurable retention time (7 – 35 days). A minimum storage amount of 1 x the Data Size (generally sufficient for up to 7 days of backups) is provided at no charge but anything consumed above this is chargeable



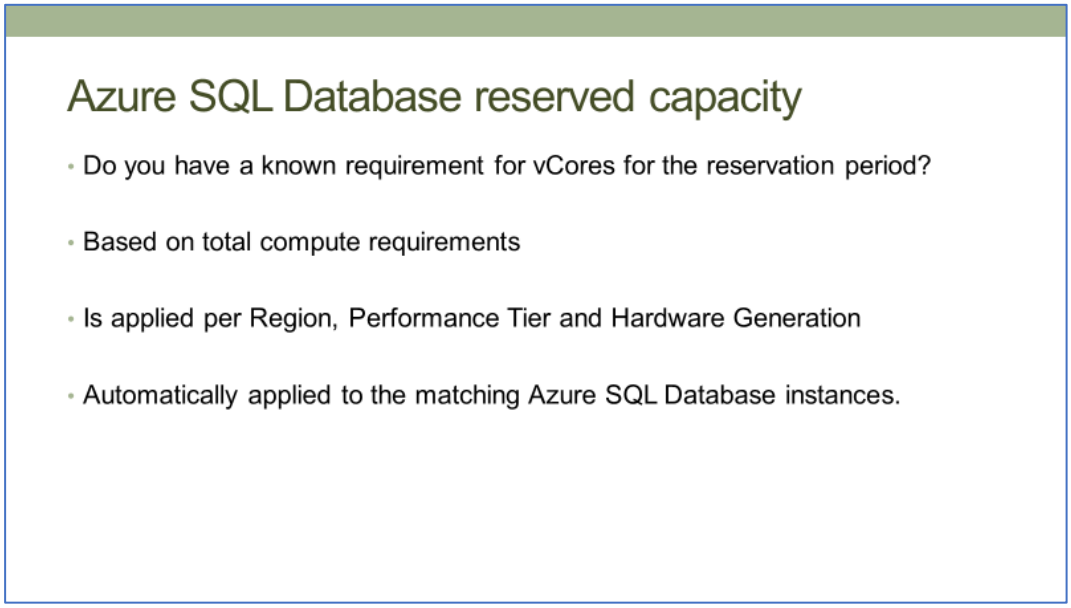
It is possible to move your Azure SQL Database between a DTU performance tier and a vCore performance tier and vice-versa. Note that you can move into and out of Hyperscale in the same manner because of the architectural differences.

There are some caveats to bear in mind – some basic such as the physical size of the Azure SQL Database must fit into the restrictions of the destination performance tier. Others are more specific to the use case that you have, e.g.

- If using Azure Hybrid Benefit, you can also move to the Business Critical performance tier if you have Enterprise Edition licenses to migrate
- When using Geo-Replication, the order of migration between tiers for the primary and secondary databases is important (see https://docs.microsoft.com/en-us/azure/sql-database/sql-databaseservice-tiers-vcore#migration-from-dtu-model-to-vcore-model)

Changing the Tier can be done in a number of ways:

- Azure portal
- PowerShell
- Azure CLI
- T-SQL
- REST API



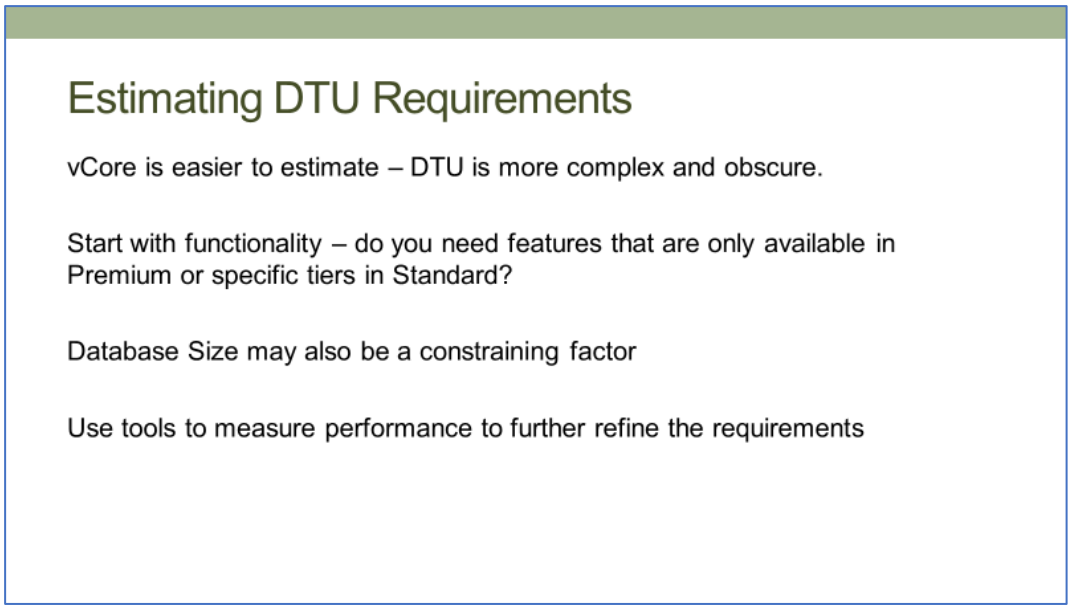
Microsoft now offers the option to purchases “Reserved Capacity” for Azure SQL Databases using the vCore Performance Tiers.

NOTE – Reserved Capacity does not exclusively reserve the physical resources but instead provides a discounted price in return for a commitment to use the resources for a period of 1 or 3 years. At the end of the period the price will revert to Pay-AsYou-Go as Reserved Capacity does not auto-renew – you will need to purchase it again and reassign the resources.

For example, let's suppose that you are running one general purpose, Gen5 – 16 vCore elastic pool, and two business critical, Gen5 – 4 vCore single databases. Further, let's supposed that you plan to deploy within the next month an additional general purpose, Gen5 – 16 vCore elastic pool, and one business critical, Gen5 – 32 vCore elastic pool. Also, let's suppose that you know that you will need these resources for at least 1 year.

In this case you should purchase a 32 (2x16) vCores, 1 year reservation for SQL Database Single/Elastic Pool General Purpose - Compute Gen5 and a 40 (2x4 + 32) vCore 1 year reservation for SQL Database Single/Elastic Pool Business Critical - Compute Gen5.

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



The vCore model provides a very straight forward way to estimate the Performance

Tier that will be required for your migrated databases. It is far easier to compare the CPU number, type and performance from your on-premises or VM SQL Server and map that to the corresponding vCore Performance Tier.

With the DTU model, this is not anywhere near as straight forward.

The starting point should be based on the required functionality and whether this is available at the DTU Performance Tier that you want. For example, columnstore indexes require a minimum of a Standard S3 performance tier. If you have a need for columnstores with a low to medium overall capacity the a S3 may be the easy choice.

The size of the database may also be a defining factor. Note the previous table with the capacity limits per performance tier and identify whether you are constrained by the size limitations.

Where you are constrained to e.g. the Premium Tier then then ease of movement between levels in the tier make it easy to initially over-estimate the requirements and through monitoring of the DTU usage adjust automatically using Azure Automation to set the appropriate performance level.

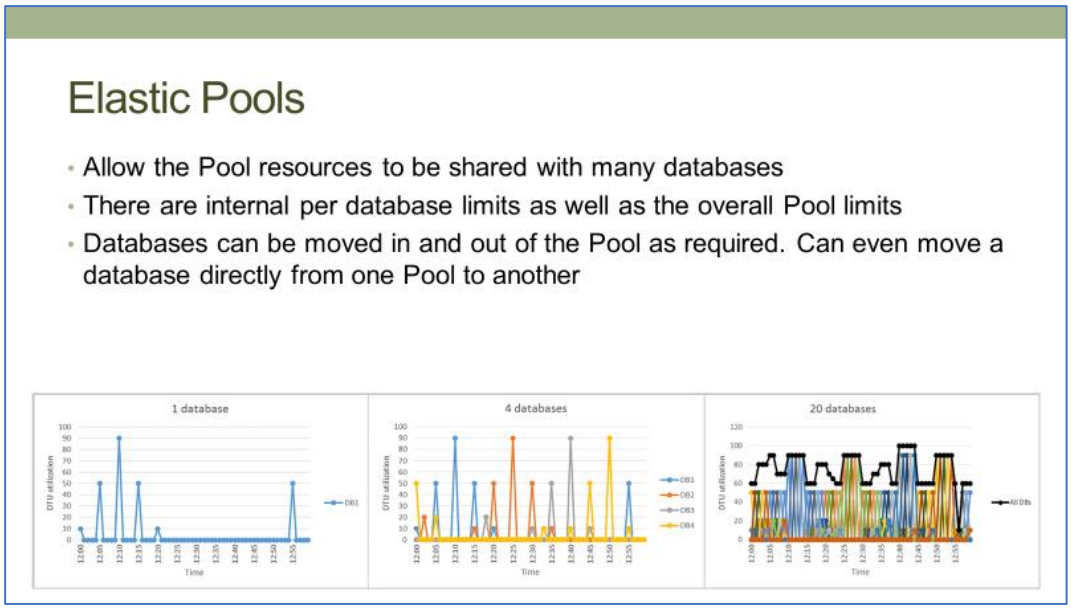
There are tools that can help with the estimation – The Azure SQL Database DTU Calculator (https://dtucalculator.azurewebsites.net/) can help determine the appropriate DTU performance tier to start from – do not expect that this will be a set and forget as further monitoring and optimisation of your Azure SQL Database will likely result in some movement between performance tiers.

The tool works well when there is only a single database in use in your SQL Server since it is based on the recording of the PerfMon Counters:

- Processor - % Processor Time
- Logical Disk – Disk Reads/sec
- Logical Disk – Disk Writes/sec
- Database – Log Bytes Flushed/sec

If you are able to isolate the individual database metrics using the DMVs and generate output in the required CSV format, then you can still use the analysis part of this tool to estimate the individual DTU requirements.

Microsoft's Data Migration Assistant (https://docs.microsoft.com/enus/sql/dma/dma-sku-recommend-sql-db?view=sql-server-2017) can also provide a recommendation for the minimum Azure SQL Database SKU that will be appropriate for your workload. This is only available when the source SQL Server instance is SQL Server 2016 or later.



What Are Elastic Pools?

When we looked at the resources (vCores/DTU) previously, these were defined at the individual Azure SQL Database level. If your database has a steady level of demand then this is a good fit as it ensures that the required resources are always available.

However, what happens if your database access pattern has periodic bursts of activity followed by periods of little use? In this scenario you may feel that you are paying for capacity that you rarely use and the cost benefits of Azure SQL Database are significantly reduced. Conversely, if you reduce the Performance Tier to be more reflective of your general usage pattern then you may experience performance issues whenever the usage ramps up.

Elastic Pools provide a possible solution to this dilemma. Elastic Pools allow you to run multiple, isolated and independent Azure SQL Databases within a private pool of resources dedicated to just you. Instead of having to define each individual Azure SQL Database resource tier, you can define the resources available to be shared amongst all databases in the Elastic Pool along with per-database restrictions as well.

As these are an extension to the Azure SQL Database service, you get all the same benefits in terms of SLA, HA replica support, geo-replication, point-in-time restores etc.

Why Use Elastic Pools?

Depending upon your usage patterns for your individual databases, an Elastic Pool may provide a far better ROI. The type of database that generally benefits from being placed in an Elastic Pool is one which has low average consumption and infrequent spikes. When these databases are combined the spikes will tend to even out over time to give an overall utilisation of the Elastic Pool resources that is more consistent. A general rule of thumb is that a database should be considered for a pool when its peak utilization is about 1.5 times greater than its average utilization.

The price of a pool is a function of the pool eDTUs. While the eDTU unit price for a pool is 1.5x greater than the DTU unit price for a single database, pool eDTUs can be shared by many databases and fewer total eDTUs are needed. These distinctions in pricing and eDTU sharing are the basis of the price savings potential that pools can provide.

Choosing the Pool Size

There are a number of physical restrictions on an Elastic Pool that may be the initial defining factor for your Elastic Pool size. Some general limits include:

- Total Storage per Pool
- Maximum number of Databases per Pool
- Maximum concurrent sessions and requests per Pool
- Maximum eDTUs per Database
- Maximum Storage per Database

The above should provide you with a baseline Performance Tier. This should then be compared to the resource usage per Azure SQL Database over a representative timeframe. This is shown on the slide where the pattern for each Azure SQL Database is superimposed on top of each other with concurrent peaks being cumulative. This will provide an indication of the total Elastic Pool resources required.

Combining the information from both sets of planning will confirm the required Tier. As mentioned, the cost per eDTU is 1.5x more than the DTU price, and therefore the final step is to compare the overall pricing for the Elastic Pool with the cumulative pricing for the individual Azure SQL Databases. Where the usage patterns match the low average utilisation with infrequent spikes then it is possible to see a cost reduction with as few as 2 S3 Azure SQL Databases.

Manual Estimation Steps

1. Estimate the eDTUs or vCores needed for the pool as follows:

   For DTU-based purchasing model:
   MAX(<Total number of DBs X average DTU utilization per DB>,
   <Number of concurrently peaking DBs X Peak DTU utilization per DB)

   For vCore-based purchasing model:
   MAX(<Total number of DBs X average vCore utilization per DB>,
   <Number of concurrently peaking DBs X Peak vCore utilization per DB)

2. Estimate the storage space needed for the pool by adding the number of bytes needed for all the databases in the pool. Then determine the eDTU pool size that provides this amount of storage.

3. For the DTU-based purchasing model, take the larger of the eDTU estimates from Step 1 and Step 2. For the vCore-based purchasing model, take the vCore estimate from Step 1.

4. See the SQL Database pricing page and find the smallest pool size that is greater than the estimate from Step 3.

5. Compare the pool price from Step 5 to the price of using the appropriate compute sizes for single databases.

Using Historical Resource Usage to Estimate

The Azure Portal allows you to undertake a "What-If" scenario with your Azure SQL Databases and Elastic Pools to get sizing advice before committing your changes.

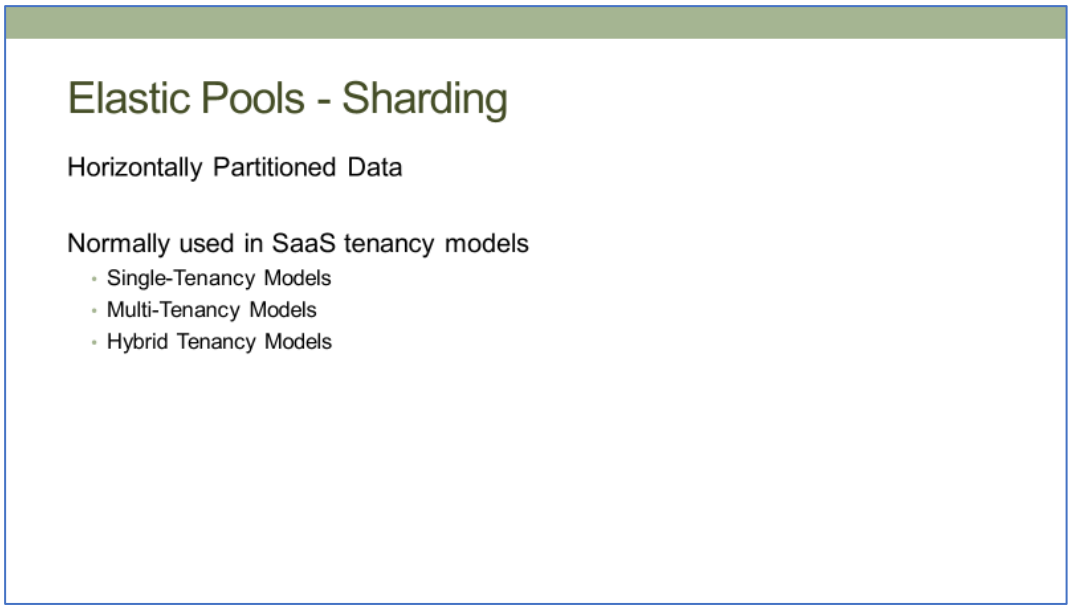
Moving Databases In/Out of Pools

It is easy to move databases in and out or even between Elastic Pools. They will take on the restrictions defined for Per Database limits of the Elastic Pool that they move into, which maybe more or less than the current limits. This is something to bear in mind as you move databases around to better balance your overall resource utilisation.

Although moving the databases only takes a few minutes, as the databases are moved they remain online and fully accessible until the very last moment, at which point any open connections are closed. As long as you have some retry logic, clients will then connect to the database in the new pool.

A sample PowerShell command to move an Azure SQL Database either into an Elastic Pool or to a different Elastic Pool is shown. The parameters for the ***Set-AzSqlDatabase*** cmdlet define the desired target, therefore to move from an Elastic Pool to stand-alone simply replace "-ElasticPoolName" with "-RequestedServiceObjectiveName"

```
Set-AzSqlDatabase `
       -ResourceGroupName myRG `
       -ServerName 'mySQLServer' `
       -DatabaseName 'myDB' `
       -ElasticPoolName 'myPool'
```



The other common use case for Elastic Pools is in conjunction with Sharding.

What is Sharding?

Sharding is a design concept where the total set of data is scaled out into multiple databases. Generally, the data is partitioned so that each customer's data is either in a dedicated database or uniquely identified in a common database through the use of a tenant identifier.

The other common feature of a Sharded configuration is the inclusion of a Catalog database to map the tenant identifiers to the correct database URIs. When combined with the Elastic Client Library, the application code can easily connect to the correct database and there is also provision for a number of management functions such as splitting a tenant from a shared database to a dedicated one and vice versa.

Sharding is often used in a SaaS model where you may have many "tenants" use your application. The main models are:

Single-Tenancy

In this model, there is a copy of the application per tenant with a dedicated database servicing it. This format does not make use of Sharding as the resources are dedicated and often deployed to a Resource Group per tenant. This also prevents the use of Elastic Pools. However, there is excellent isolation of the databases, ensuring both data security and also the ability to scale individually. However, as the number of tenants increases there are challenges with maintenance activities across a large number of databases.

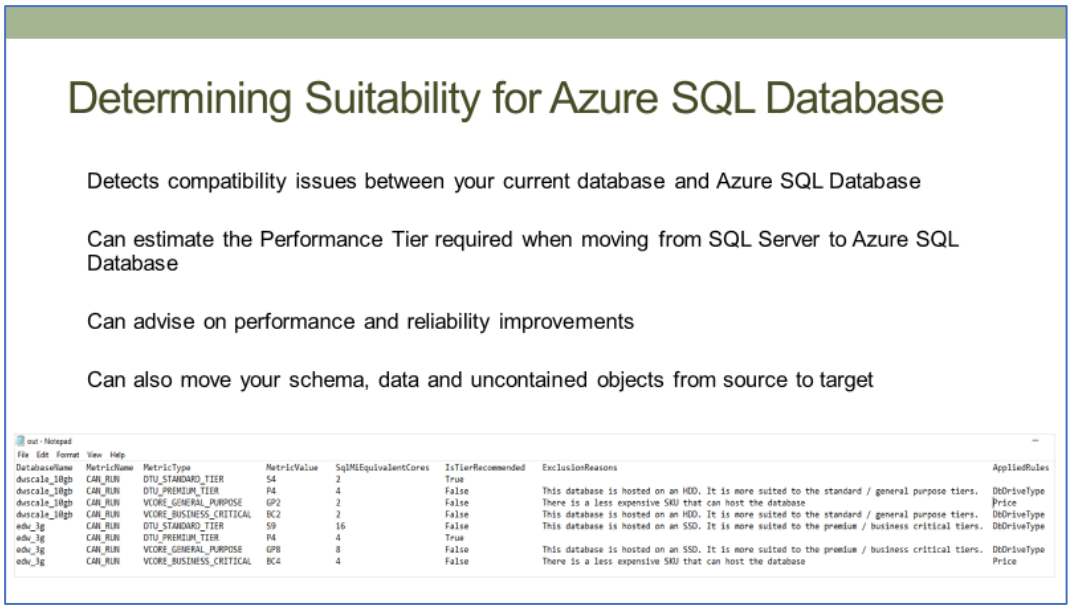
Multi-Tenancy

In this mode, there is a single shared application that can be scaled depending on the load at that level but also two options for the database layer. With a single database per tenant, the database resources can be scaled as required and also added to Elastic Pools as the individual load patterns allow. There is the advantages of still being able to undertake tasks such as restores on a tenant by tenant basis and the flexibility of moving databases in, out and between Elastic Pools. The alternative is a multi-tenanted database where the tenant identifier can be used as the Sharding key to place all the data for a tenant in a specified database. It may be that they are grouped based on the level of service purchased – e.g. all the free trial tenants in a single database to minimise costs. With the Sharding key it is still possible to move a tenant

between databases as required. Multi-tenant databases are most cost effective when there are a large number of relatively inactive tenants.

Hybrid-Tenancy

In this model, all databases have the tenant identifier in their schema and the databases are capable of storing multiple tenants' data. However. the Sharding means that effectively some of the databases may be Single-Tenanted. The hybrid model is best when there are large differences between the resource needs of identifiable groups of tenants. As the tenants demands increase they can move from different levels of shared databases up to its own new single-tenant database. Combine this with Elastic Pools and the most efficient and lowest cost model can be defined.



When considering whether Azure SQL Database is a suitable platform for your database, there are a number of steps where the Data Migration Assistant (DMA) can assist and make your job much easier.

The first step is to run the Assessment workflow to detect any cases of:

- Migration Blocking Issues- DMA will also provide recommendations to help you address these issues.
- Partially Supported / Unsupported Features – DMA provides a comprehensive set of recommendations or alternative approaches that are available.
- New Features – where the performance could benefit. These are classed in the Performance, Security and Storage buckets.

The DMA tool is a more comprehensive tool than the previous “Upgrade Advisor” tools were, and is now a complete replacement for any old versions of this tool.

Identify the Right Azure SQL Database SKU

With DMA v4.0 and later, there is now also a command line option to analyse a set of Performance Counters collected from the on-premises SQL Server machine and identify the minimum recommended Azure SQL Database performance tier. The DMA download includes a PowerShell script that can be used to gather the required Performance Counters.

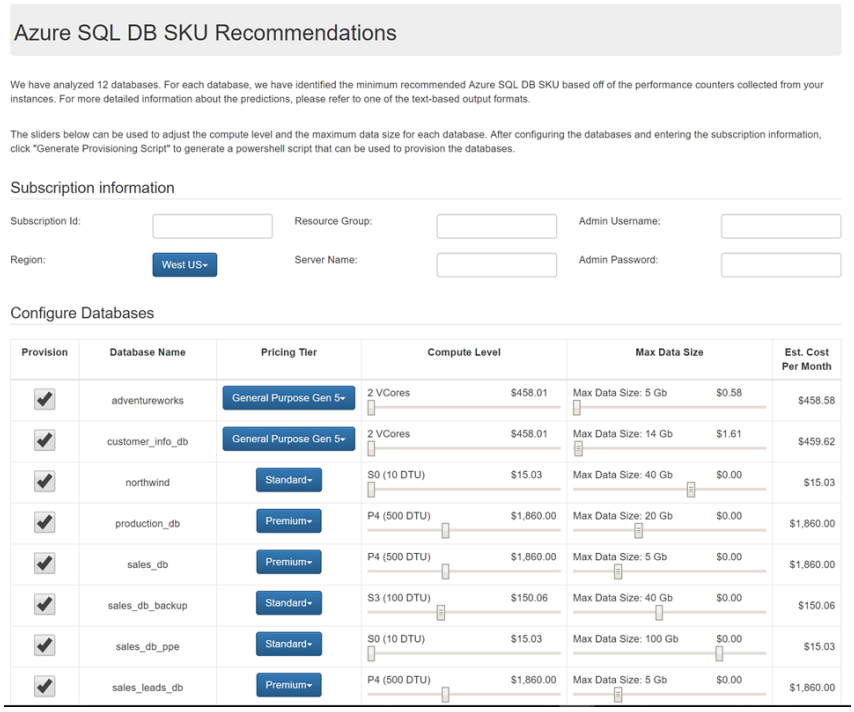
The gathering script is called using:

```
.\SkuRecommendationDataCollectionScript.ps1 `
    -ComputerName myServer `
    -OutputFilePath C:\Temp\counters.csv `
    -CollectionTimeInSeconds 600 `
    -DbConnectionString “Server=localhost;Initial Catalog=master;Integrated
    Security=SSPI;”
```

Once the file is collected then the estimator can be called using (this runs in offline mode which does not need to connect to Azure to get the current prices per region and currency):

.\DmaCmd.exe /Action=SkuRecommendation
/SkuRecommendationInputDataFilePath = “C:\Temp\counters.csv”
/SkuRecommendationOutputResultsFilePath = “C:\Temp\prices.Html”
/SkuRecommendationPreventPriceRefresh=true

A sample output is shown below.



Assess Your Migration to Azure SQL Database

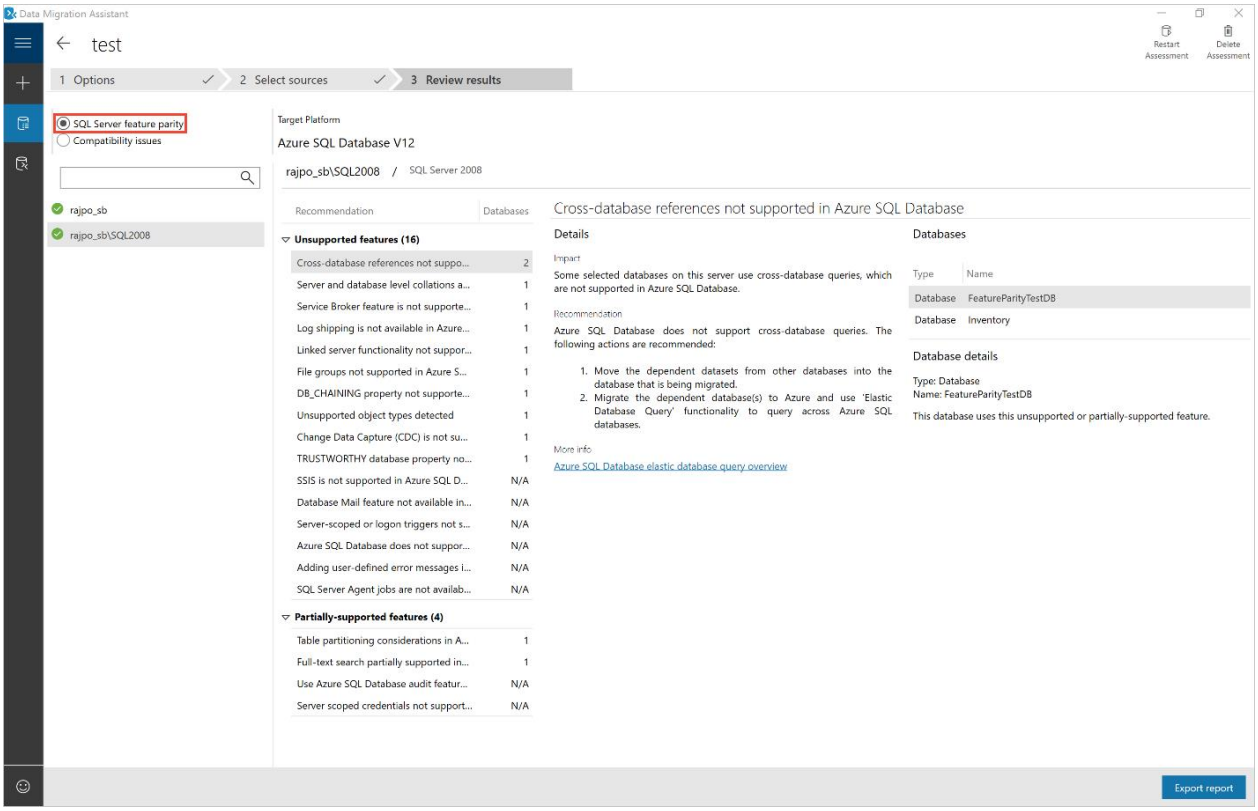
To perform an assessment, first start up the DMA GUI tool. Select New Assessment as the project type.

NOTE: at this time, the tool is unable to reload previously undertaken assessments, nor is it able to restart an assessment that stops or fails for any reason. In this case the full assessment needs to be re-executed.

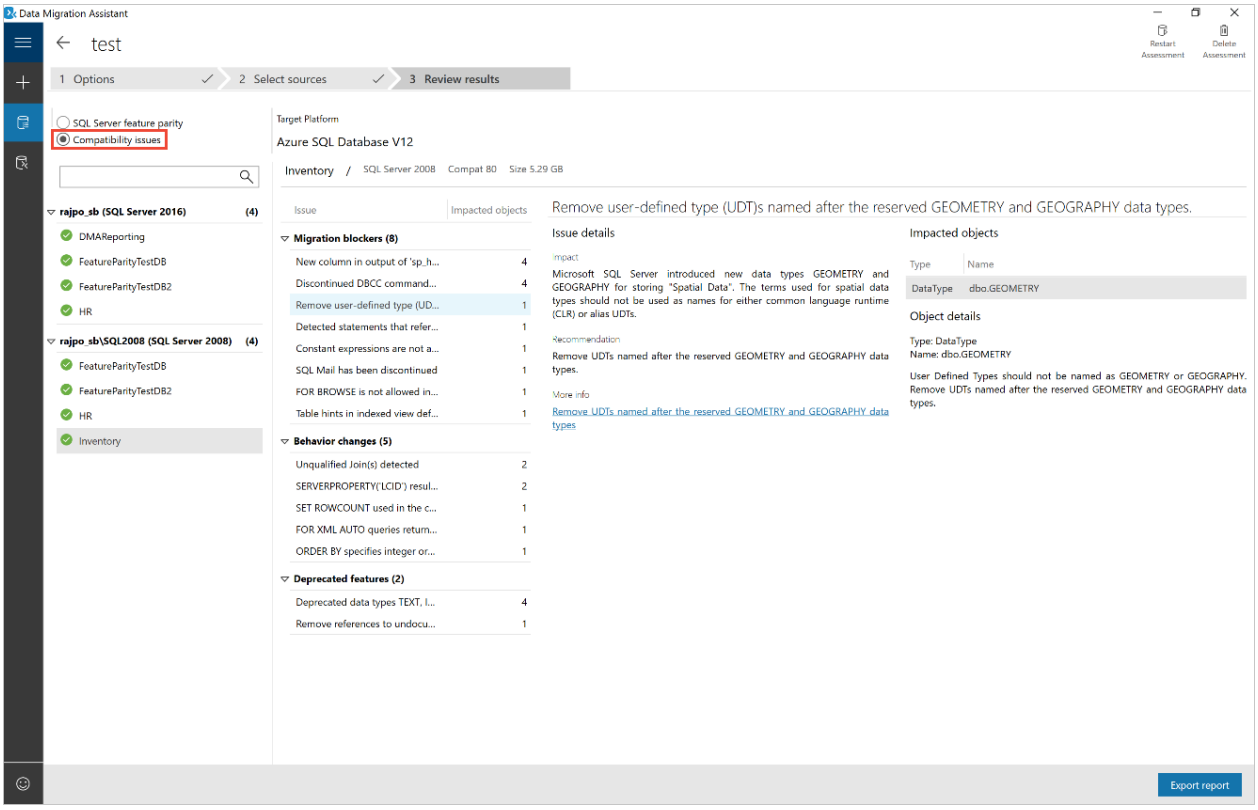
- Set the source as required and the destination to Azure SQL Database.
- Select the type of assessment to perform. You can choose one or both of “Check Database Compatibility” or “Check Feature Parity”.
- Choose the databases that you want to include in the assessment.

The duration of the assessment depends upon the number of databases and the schema size of each. The results for each database are displayed as they are completed by the DMA Tool.

In the output, the **Feature Parity** category provides a full set of recommendations and alternative approaches or mitigating steps you can use to ready the database for the migration to Azure SQL Database.



The **Compatibility Issue** category identifies any partially supported or unsupported features that will block your ability to migrate that database to Azure SQL Database. It also provides recommendations to address the blocking issues.



Consolidated Assessment Reports

By using a central tools computer, there are some additional downloads that will allow the assessments to be configured to run against multiple SQL Server instances. The results can be loaded into a data warehouse and consumed with a set of Power BI reports.

The details to configure and run this is documented at https://docs.Microsoft.com/en-us/sql/dma/dma-consolidatereports

DEMO 3: Evaluate the SKU Estimator in PowerShell – can't run the analyser but can show the HTML output and how it changes in the HTML
DEMO 4: Run an Assessment against the AdventureWorks2016 database on my local instance

Summary

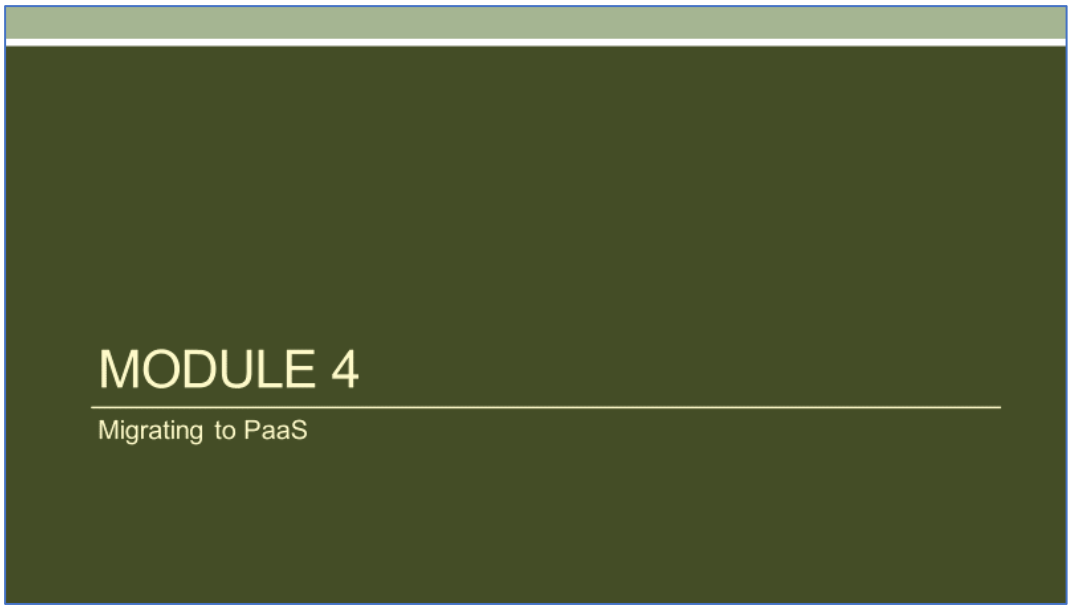
In this module we looked at the differences in performance tiers for Azure SQL Database.

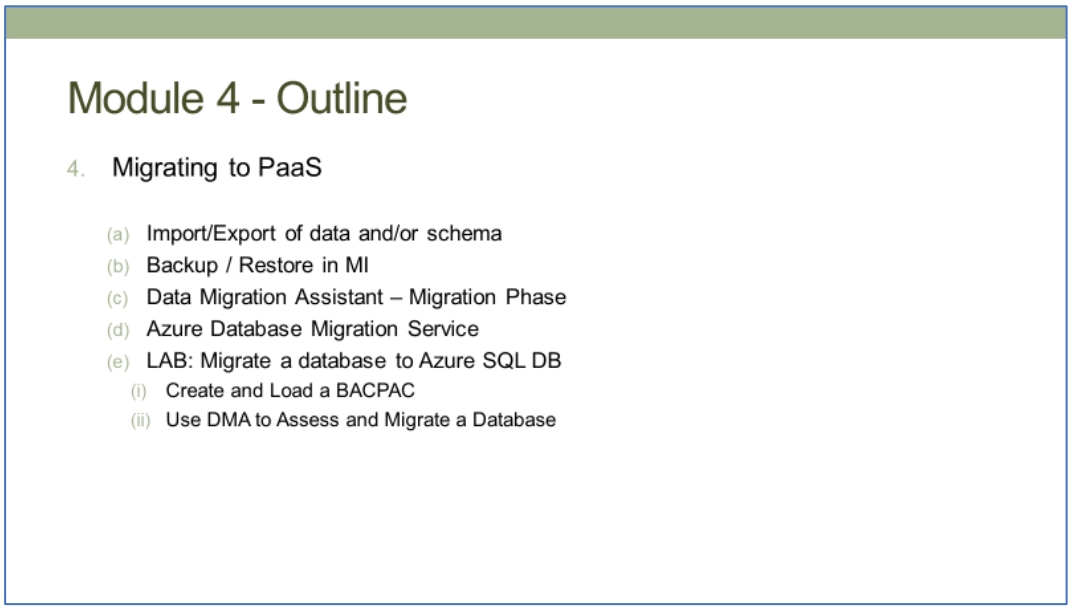
The different models (vCore and DTU) were described and compared along with some of the ways that you can make an estimation of the required target based on your current workload.

Once you have an Azure SQL Database though, it is very easy to move it between Tiers – ideally it is normally downwards to save costs, but also periodic upwards movements to handle known workload increases.

We looked at different models for helping at scale – whether using Elastic Pools to share costs between databases or by Sharding out to smaller component databases and allowing the Elastic Tools to manage the splits, merges and movements of the partitioned data.

We finally were introduced you to some of the tooling that is rolling out to make it easier for you to assess what options are right for your databases and be better placed to plan your costs.





Introduction

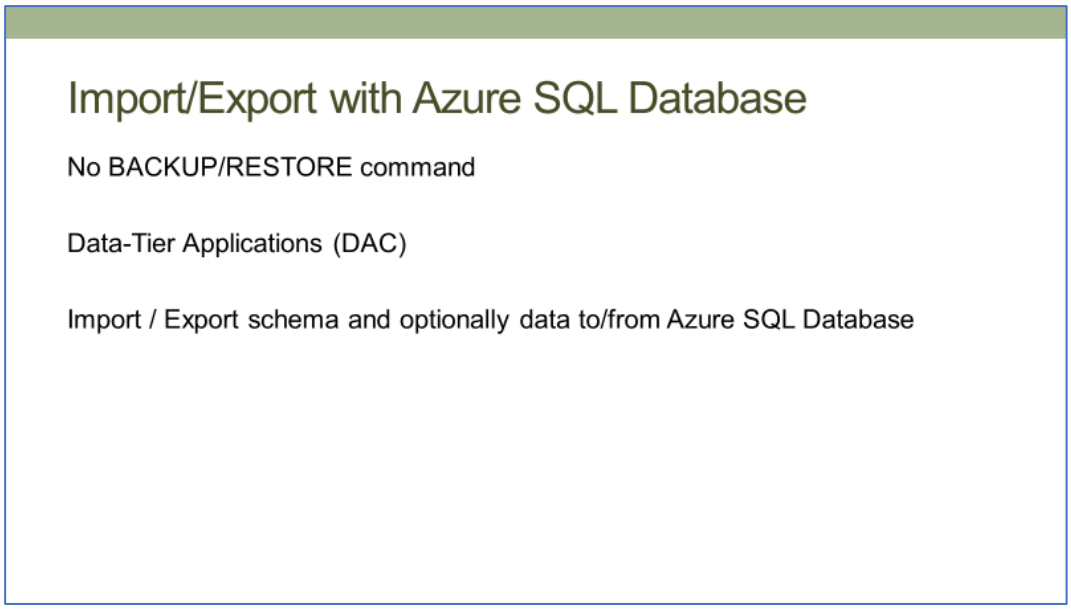
This module will cover how we implement the migration to Azure SQL Database or a Managed Instance.

We will investigate the ways that this can be undertaken, from the use of DACPAC/BACPAC packages, the ability to use a regular SQL Server backup when moving to Managed Instances, the assistance that DMA can provide to simplify the process and finally how setting up a DMS application can do this at scale and/or with minimal down-time to your applications.

There is a LAB at the end of this Module where you will move some databases to Azure SQL Database using a basic BACPAC and also using DMA to simplify the loading.

Resources

DACPACs and BACPACs:	https://docs.microsoft.com/en-us/sql/relational-databases/data-tier-applications/data-tier-applications?view=sql-server-2017 (NOTE: references to versions are pretty messed up in the docs)
Deploying DACs:	https://docs.microsoft.com/en-us/sql/relational-databases/data-tier-applications/deploy-a-database-by-using-a-dac?view=sql-server-2017
Import BACPAC to Azure SQL Database:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-import
Sqlpackage.exe	https://docs.microsoft.com/en-au/sql/tools/sqlpackage?view=sql-server-2017
Restore to Managed Instances:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-managed-instance-get-started-restore
Data Migration Assistant:	https://docs.microsoft.com/en-us/sql/dma/dma-overview?view=sql-server-2017
Database Migration Service:	https://docs.microsoft.com/en-us/azure/dms/dms-overview
Microsoft Data Migration Blog:	https://techcommunity.microsoft.com/t5/Microsoft-Data-Migration/bg-p/MicrosoftDataMigration
DMS Online Migration Limitations:	https://docs.microsoft.com/en-au/azure/dms/known-issues-azure-sql-online
Supported DMS Scenarios:	https://docs.microsoft.com/en-us/azure/dms/resource-scenario-status#migration-scenario-support



As we have learned, Azure SQL Database does not expose the standard SQL Server backup and restore formats. The backup process is fully automatic and invisible to us as users and administrators (although we can add additional long-term backups as we'll see in Module 5).

So, if we cannot restore a backup from a SQL Server database, how can we deploy our database to Azure SQL Database?

Data Import / Export Options

Data-Tier Applications (with their awful acronym – DAC) are effectively a logical entity that defines all of the SQL Server objects within a database. It includes all the tables, views, stored procedures, logins and users. A DAC forms a self-contained unit of deployment and is portable through the generation of a DACPAC.

If the data is also exported and included with the artefact then the entire package is a BACPAC. A BACPAC also includes the same schema definition objects that a DACPAC would contain.

Tools such as SSMS encapsulate the DAC wizards, which allow the DBA or Developer to easily export their SQL Server or Azure SQL Database stored database to a BACPAC/DACPAC. One of the major benefits of the DAC however is that there is also an upgrade process which will compare the BACPAC/DACPAC with the currently deployed database and advise the user of the risk of any data loss as well as providing an upgrade plan which can be reviewed before progressing. The command line executable that performs the DAC functionality is included with a SQL Server installation and is named ***sqlpackage*** and can be downloaded and run on Windows, Linux or macOS.

The DACPAC produces a packaged set of XML files that fully describe the schema and relationships of the database objects. Depending upon the version of the DACFx framework used to perform the export, the data component of a BACPAC will either be in compressed JSON (older format) or native format BCP (newer format).

An alternative approach that provides a similar outcome but with more manual effort and more control is to use the Script option to produce T-SQL scripts for the database objects. When using this option for a SQL Server database, it is important to ensure that in the Scripting Options, "Match script settings to source" is set to "False" and "Script for the database engine type" is set to "Microsoft Azure SQL Database". This will produce the schema – to get the data we can

use the BCP utility to export the data for each of the tables we are interested in. To load this, we run the script to create the objects and then use BCP again to load the data to the tables.

Note that as the BACPAC data extract is native BCP format, it is also possible to extract the files from the archive and use BCP to selectively load the tables.

Sample Command Line Options

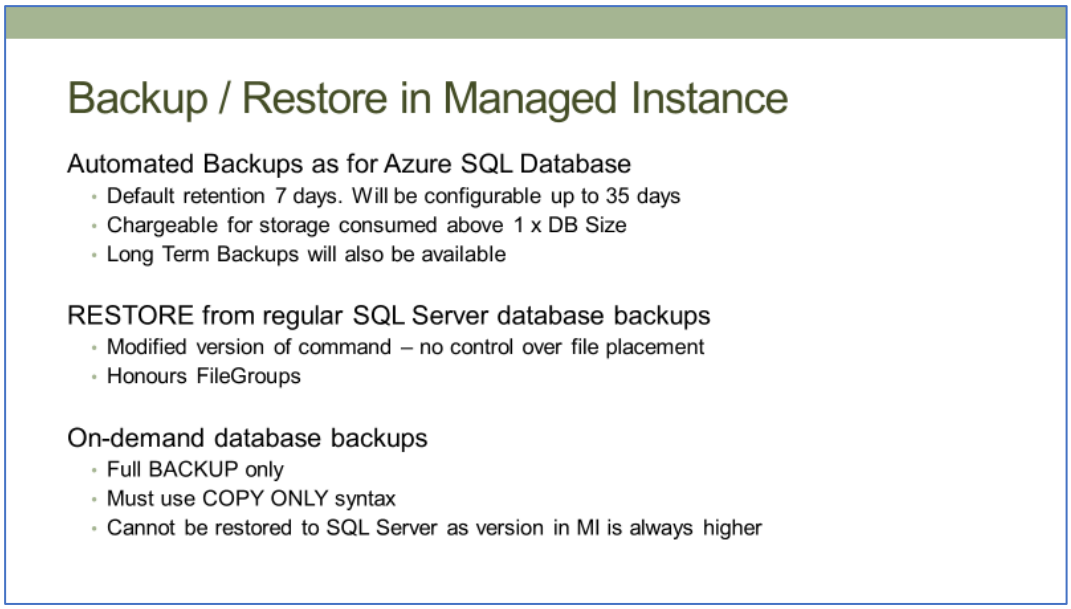
To export to a BACPAC file using the sqlpackage command line:

1) Open a command prompt as Administrator and navigate to c:\Program Files (x86)\Microsoft SQL Server\<version number>\DAC\bin
2) Run: sqlpackage.exe /Action:Export /SourceDatabaseName:MyDB /SourceServerName:MySQLServerInstance /TargetFile:MyOutputPath /OverwriteFiles:True

To import a BACPAC using PowerShell, the syntax would be like the following. Note that to use the PowerShell cmdlets for import/export the files must be in an Azure Storage Account:

```
# Import bacpac to database with an S3 performance level
$importRequest = New-AzureRmSqlDatabaseImport `
        -ResourceGroupName $resourcegroupname `
        -ServerName $servername `
        -DatabaseName $Database `
        -DatabaseMaxSizeBytes "262144000" `
        -StorageKeyType "StorageAccessKey" `
        -StorageKey $(Get-AzureRmStorageAccountKey -ResourceGroupName
$resourcegroupname -StorageAccountName $StorageAccount).Value[0] `
        -StorageUri "$StorageURI$OutputFileName" `
        -Edition "Standard" `
        -ServiceObjectiveName "S3" `
        -AdministratorLogin $adminlogin `
        -AdministratorLoginPassword $(ConvertTo-SecureString -String $password -AsPlainText -
Force)

# Check import status and wait for the import to complete
$importStatus = Get-AzureRmSqlDatabaseImportExportStatus `
        -OperationStatusLink $importRequest.OperationStatusLink
Write-Host "Importing" -NoNewline
while ($importStatus.Status -eq "InProgress")
{
        $importStatus = Get-AzureRmSqlDatabaseImportExportStatus `
                -OperationStatusLink $importRequest.OperationStatusLink
        Write-Host "." -NoNewline
        Start-Sleep -s 10
}
Write-Host ""
$importStatus
```



Azure SQL Database Managed Instances allow additional flexibility in the use of regular SQL Server backups. You are able to restore a backup taken from any version of SQL Server from 2005 onwards to a Managed Instance, however there are some restrictions as a result of Microsoft managing the underlying platform.

Automated Backups

The automated backup process is the same as that for Azure SQL Database. The exact timings of the backups are managed by the Managed Instance based on workload, but in general there is a FULL backup weekly, DIFFERENTIAL backups every 12-24 hours and TRANSACTION LOG backups every 5-10 minutes.

The retention is configurable between 7 and 35 days. Long Term backup retention will also be enabled for Managed Instance at a later date. To get around this limit for now, you could schedule your own FULL backups with COPY ONLY using a SQL Agent job.

BACKUP Command Differences

Obviously, not everything that is available in the SQL Server version of the BACKUP command will be available in the Managed Instance version. The main differences are:

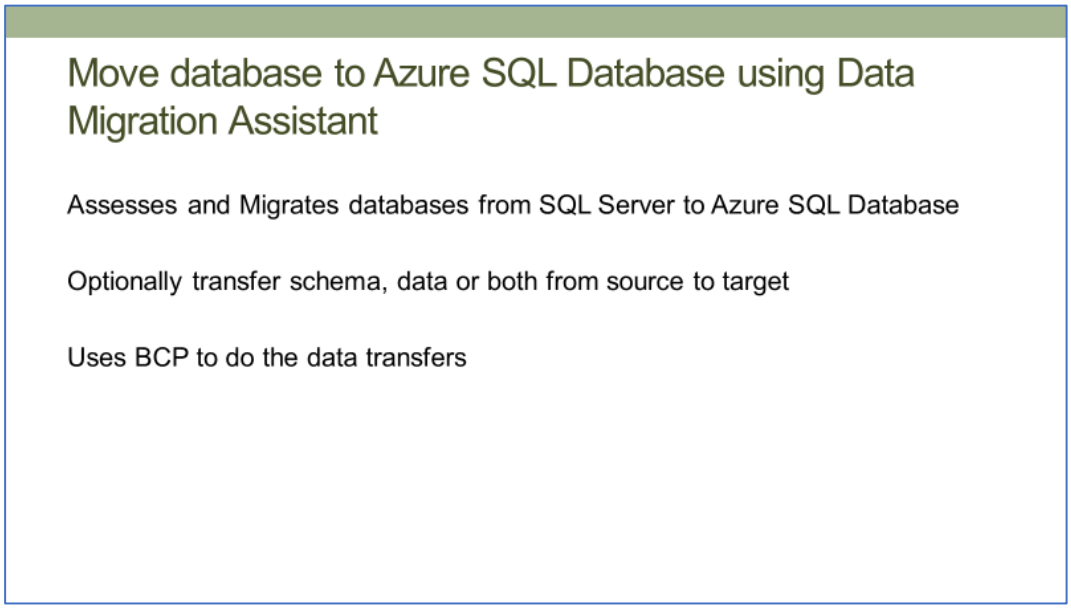
- Can only do a FULL backup. Taking DIFF or TLOG backups is not permitted
- Must include the WITH COPY ONLY switch
- Destination must be Azure Blob Storage. Note; their size restrictions of 32 stripes with each stripe being up to 195GB in size (see notes under RESTORE)

RESTORE Command Differences

As for BACKUPs, there are restrictions with the RESTORE command:

- Source must be Azure Storage
- Note the storage size restrictions can be overcome by BACKUP the SQL Server database on-premises then copy to Azure Storage. This will end up using a different type of blob which doesn't require the striping
- The RESTORE will continue to run even if you disconnect your client machine
- Only RESTORE DATABASE and the meta-data related RESTORE (e.g. RESTORE HEADER ONLY) options can be executed from a Managed Instance

- If the source database was Bulk-Logged or Simple recovery it will be automatically changed to Full recovery
- If the source database had auto-close enabled, this will be disabled
- If the source database did not have BROKER ENABLED then this will be enabled
- If no Memory-Optimised Filegroup existed, one will be created. If one existed but was not named XTP then it will be renamed to XTP
- Backup file cannot contain FILESTREAM data – the RESTORE will fail
- Backup file also cannot hold multiple backup sets or have databases with multiple transaction log files
- There are current restrictions on databases that have or did have In-Memory objects



We've already explored the Assessment Phase of the DMA. The second part that this tool provides the ability to undertake your migration.

Migration your Database to Azure SQL Database

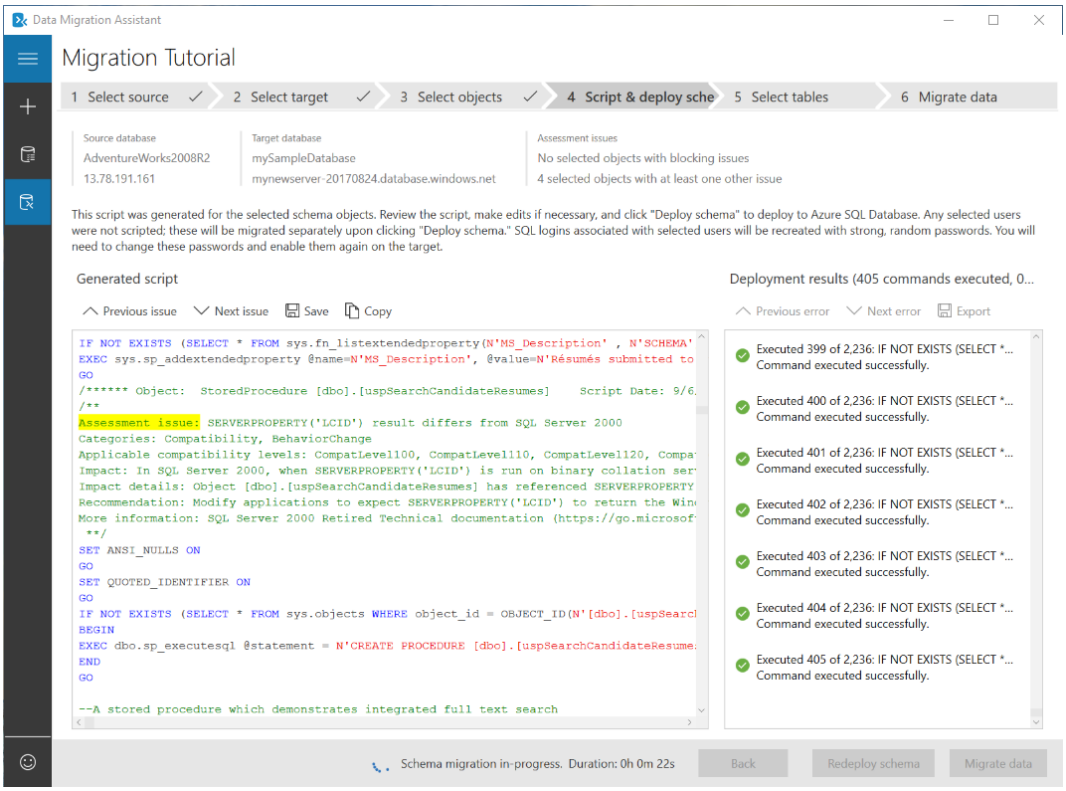
To perform a migration, first start up the DMA GUI tool. Select New Migration as the project type.

NOTE: at this time, the tool is unable to reload previously undertaken assessments, nor is it able to restart an assessment that stops or fails for any reason. In this case the full assessment needs to be re-executed.

- Set the source as required and the destination to Azure SQL Database.
- Select the type of migration to perform. You can choose one or both of "Schema" or "Data".
- Connect to the source database
- Connect to your Azure SQL Database. At this time, if you intend to transfer the schema then the Azure SQL Database needs to already exist. If you need a new empty database then create this first.

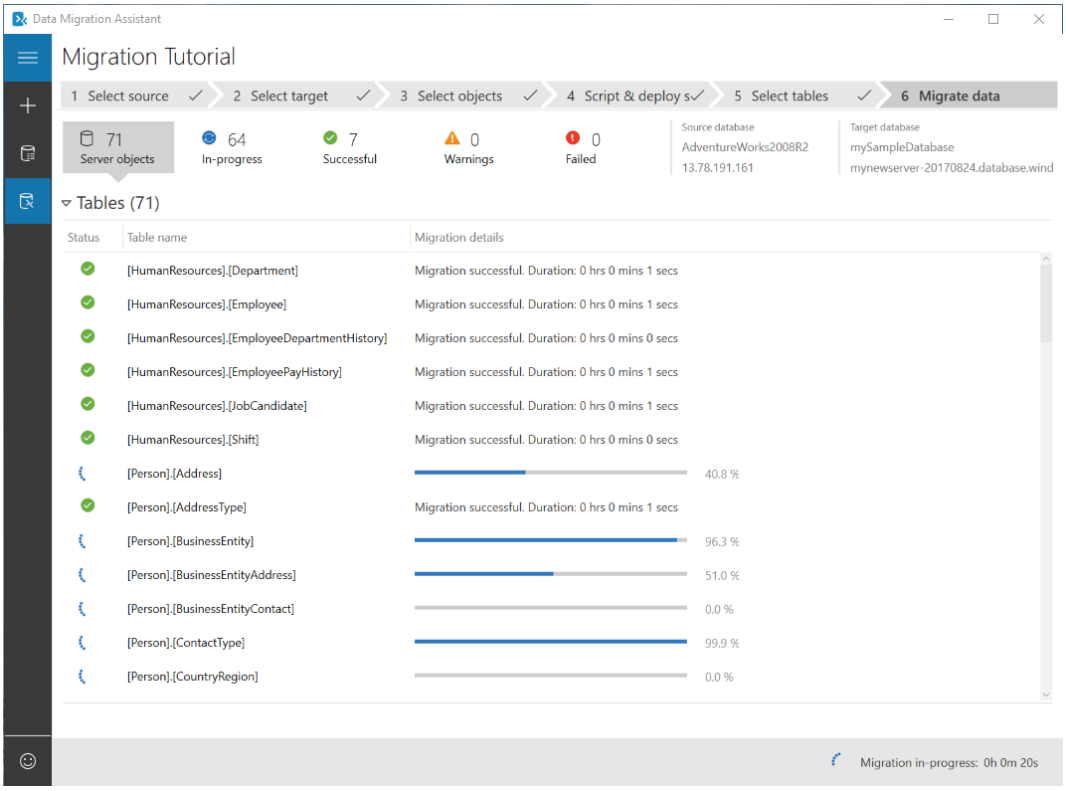
When Schema is Included:

- Select the tables to include. This generates a T-SQL script that can be executed against the target.
- Review the script and update if necessary. Users are not included in the script but will be migrated separately when the schema is deployed. Any SQL Logins associated with the Users will be recreated BUT with random, strong passwords – you will need to update the passwords to ensure your users and/or applications can connect.
- Click on "Deploy Schema" to execute the script on the target.



When Data is Included:

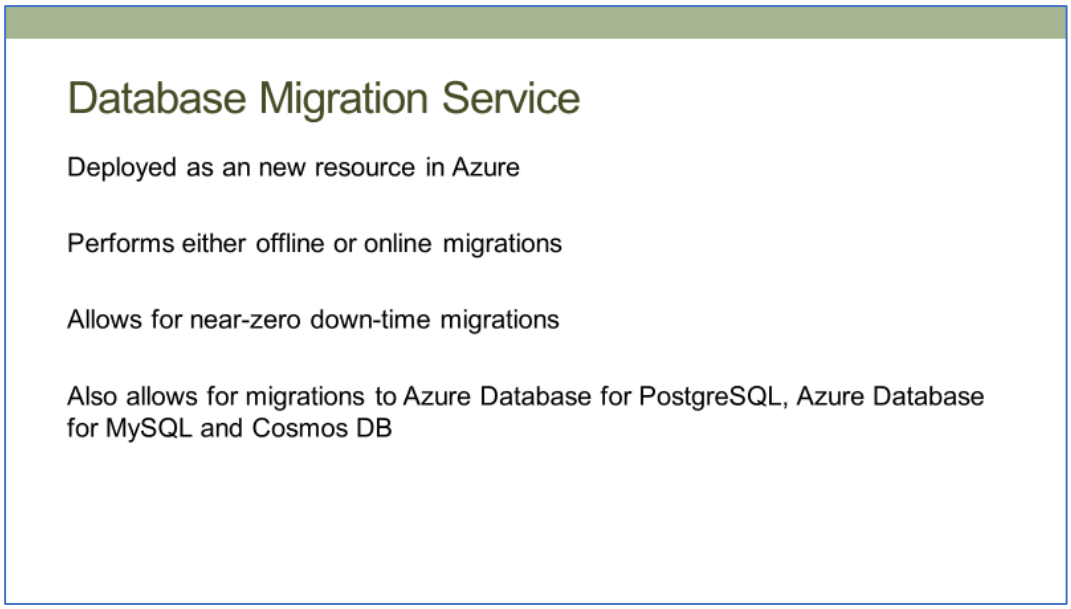
- Select the tables to include. The DMA tool will check that the table exists on the target, the schemas match and that the target copy is empty. If these conditions are met then the table can be selected for Data Migration
- Click on “Start Data Migration”. The DMA tool uses BCP under the covers to perform the data migration.



The duration of the migration depends upon the size of the data in each table. The results for each table are displayed as they are completed by the DMA Tool.

DEMO 5: Prepare an empty SQL Database to receive the migration

DEMO 6: Use DMA to migrate the MyTest database from my local instance to the SQL Database created above



The Data Migration Service (DMS) is a useful tool for migrating large size databases or large numbers of databases to Azure SQL Database or Managed Instance. It runs in one of two modes - either for offline migrations (effectively to a fixed point in time – useful if the source database can be taken offline from the application for the duration) or online where the target is kept in sync with the source up to the point where you decide to cut-over to use the new Azure database.

DMS to Azure SQL Database Prerequisites

Before you can make use of DMS, there are a number of aspects to consider. Some are general requirements, but there are also some specific requirements for the different types of migration:

General Prerequisites

- Source must have TCP/IP protocol enabled
- DMS must be created in a VNET and the on-premises SQL Server must be able to access the VNET using Express Route or VPN
- The VNET NSG must allow ports 443, 53, 9354, 445, 12000 and 1433
- Azure SQL Database firewall rules must permit access from the DMS VNET
- Specified credentials must have CONTROL SERVER permissions on the source and CONTROL DATABASE on the target

Offline Migration Prerequisites

- Source must be SQL Server 2005 or later.

Online Migration Prerequisites

- Source must be SQL Server 2005 or later or Amazon RDS
- Source databases must be either Full or Bulk-Logged recovery model
- A Full Backup of the databases must have been taken before attempting migration
- If any table does not have a Primary Key then CDC must be enabled for the database and table
- The source SQL Server must have the replication distributor role enabled
- The target Azure SQL Database must have database triggers disabled

Preparing for DMS to Azure SQL Database

Once the prerequisites have been satisfied, the next step is to use the DMA tool to complete an Assessment of the source databases. Once the remediation steps for any issues have been completed, again use DMA to migrate the *Schema Only* of the source databases to the new Azure SQL Database.

Configure and Run DMS

In the Azure Portal, create an instance of Database Migration Service. It is recommended to create the resource in the same region as the target Azure SQL Database to minimise data movement. The Pricing Tiers can be selected depending upon whether online or offline migrations will be used. An online migration will require greater resources.

Note that if you select the option to create a new VNET during the creation of the DMS, it does not allow you full control to configure the VNET at this time. You will need to go back to the VNET and adjust the settings accordingly, including any NSG and IP Address ranges.

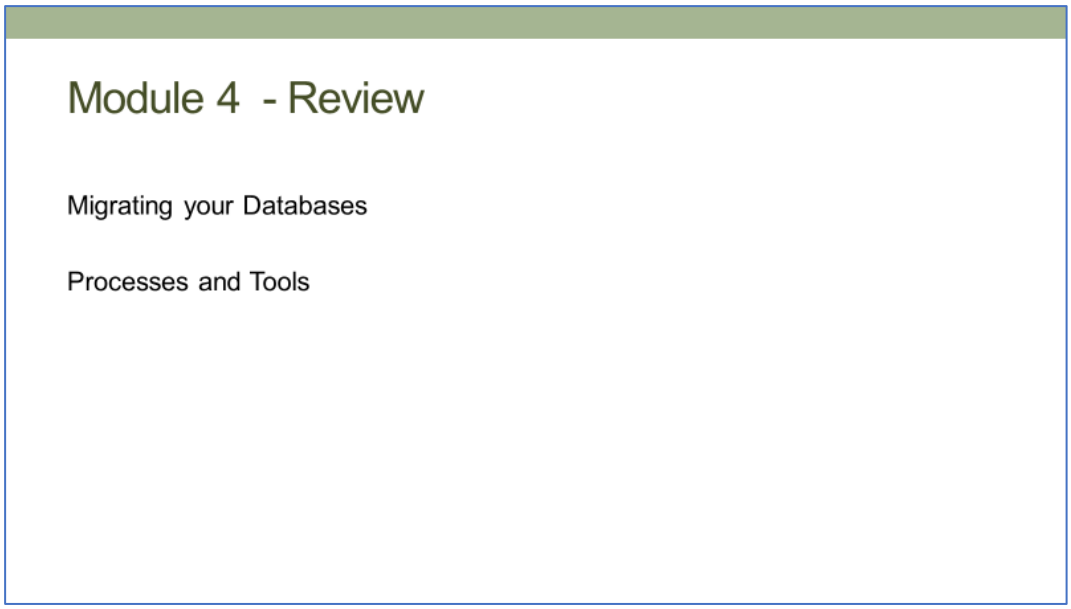
Create a new Migration Project. When you enter the source SQL Server instance the service will confirm that it is able to connect. If it cannot then you will need to troubleshoot the various linkages to ensure that it can reach your server either over Express Route or your VPN. You may have to adjust the Windows Firewall on your SQL Server to ensure that the IP Address of the DMS VNET is permitted. The same check occurs for the target Azure SQL Database.

Once connected you can select the databases to migrate and then for each database select the tables. Empty tables on the target are selected by default – if you wish to re-migrate the data for any populated table then you need to select them explicitly. The data in these tables will be deleted before the migration commences.

The migration can then be run.

For an offline migration, once the Status of the migration is shown as *Completed* then the process is complete and your Azure SQL Database is ready for use. For an online migration, once the initial *Full Load* component has completed, the database is marked as *Ready to Cutover*. When ready to perform the cutover, you'll need to stop the incoming transactions on the source database and wait until the *Pending Changes* counter has reduced to 0. You can then select the box to confirm the cutover and apply. Once the status is shown as *Completed* you can switch your application connections to the Azure SQL Database and re-enable the applications.

Refer to https://docs.microsoft.com/en-au/azure/dms/known-issues-azure-sql-online for the current restrictions associated with online migrations.



Summary

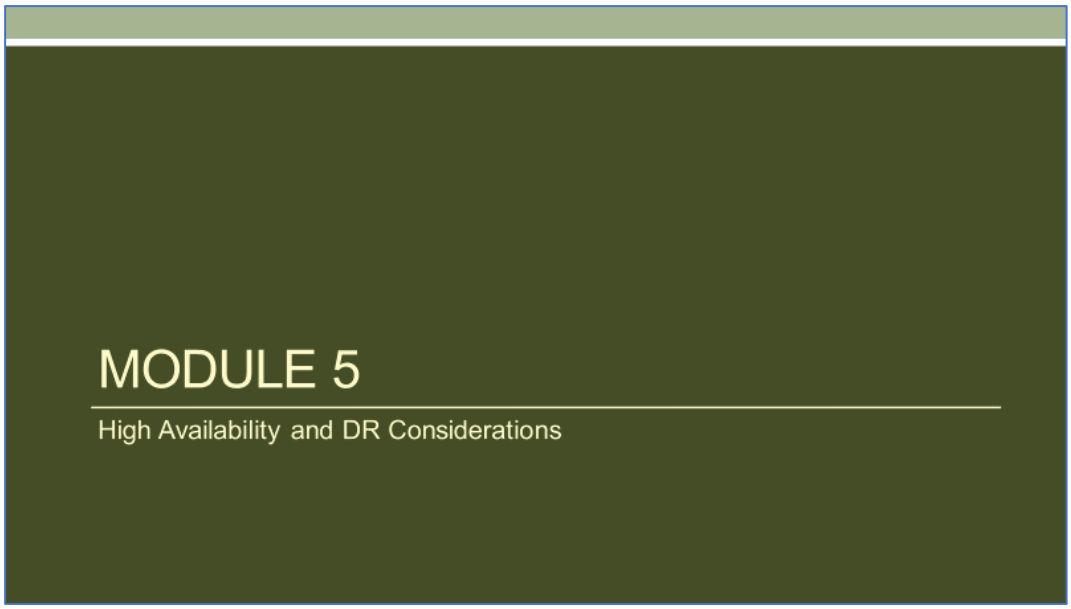
In this module we started moving databases to Azure SQL Database.

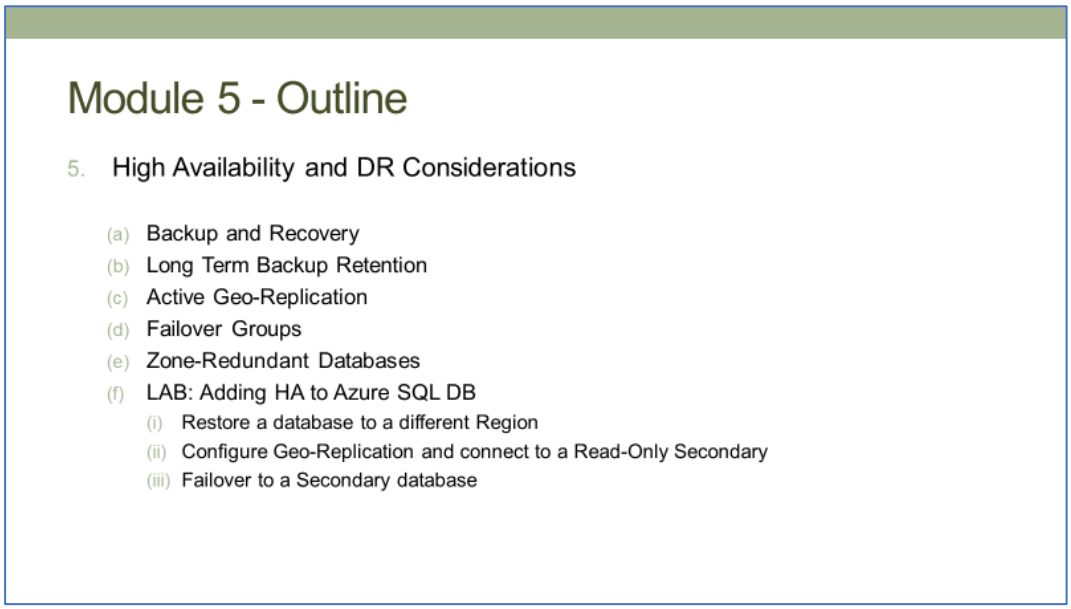
We looked at the options for offline and online migrations with near-zero downtime for the applications.

With BACPACs we can easily export/import, and importantly this is the one way to get the database and data back out from Azure SQL Database as we can't get backups out and if we could they couldn't restore on premises as the version would be too high.

Managed Instances are easier to move to in so far as they allow a native backup to be restored, but again they don't restore outside again.

Our tooling is again becoming more thorough in this space. DMA allows us to do a database by database migration but the DMS service can do this at scale – however it is complex to set up and configure as the documentation is a bit behind the product.





Introduction

This module will cover the built-in and configurable options for ensuring the appropriate HA and DR requirements.

We will investigate the various options that are available to add in additional redundancy and ensure recovery of your databases with the minimum amount of potential data loss and administrative effort.

There is a LAB at the end of this Module where you will perform some of the main HA and DR activities.

Azure SQL Database does a great job on data integrity:

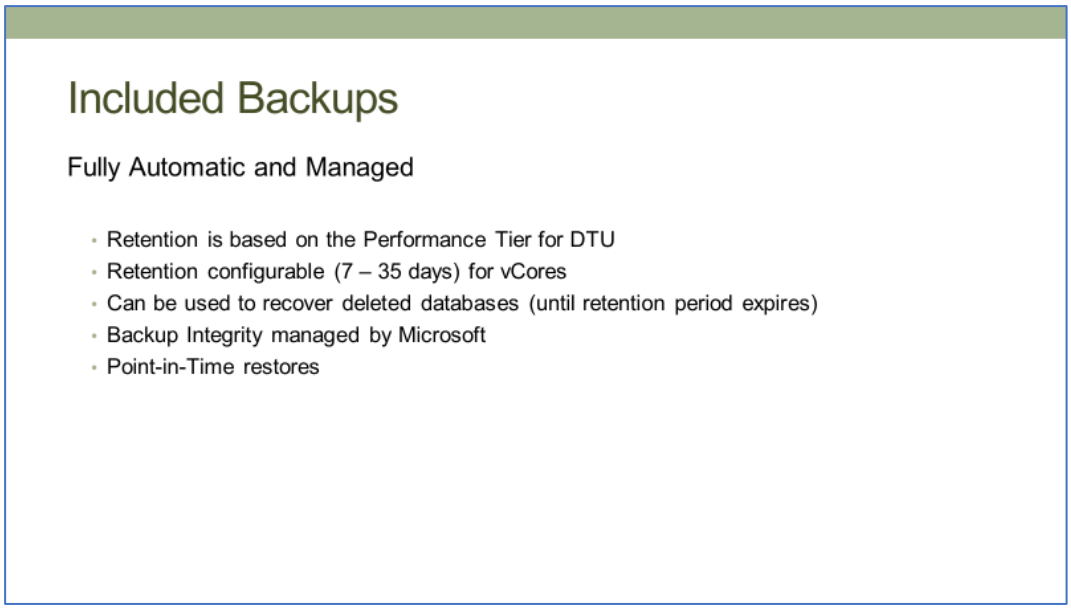
1) Error alert monitoring … if the SQL Server process throws any errors, these are routed directly to Azure engineers for investigation
2) Backup/Restore integrity check – Azure engineers automatically run selected test restores. Any issues raise an alert to engineers who may contact you if action is required.

3) IO Lost Write detection … This tracks page writes and their associated LSN. On read, the expected LSN is compared to the actual. If no match, engineering team immediately notified
4) Automatic Page Repair – the multiple copies of your DB allows for transparent replacement of a corrupted page with a copy from one of the replicas with no data loss or downtime. With the architecture (see later in this module) Basic, Standard and General Purpose tiers need to be using geo-replication to enable this.
5) Page Verify Checksum is always set. TCP/IP also uses transport level checksums.

If they can solve it transparently they will – you'll never know. If not, they will contact you ASAP and work with you to resolve. This may be as easy as Azure doing a point-in-time restore of your DB to just before the corruption and making that available to you for free. So, with all this, you don't have to run DBCC CHECKDB, but if you are paranoid (like any good DBA) then you might choose to do so.

Resources

Data Integrity in Azure SQL Database:	https://azure.microsoft.com/en-us/blog/data-integrity-in-azure-sql-database/
Ensuring Backup Integrity:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-automated-backups#how-does-microsoft-ensure-backup-integrity
Virtual Networks and Failover Groups:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-geo-replication-overview#failover-groups-and-network-security
Long Term Retention:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-long-term-backup-retention-configure
Geo-Replication:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-geo-replication-portal
Azure Availability Zones:	https://docs.microsoft.com/en-us/azure/availability-zones/az-overview
Azure SQL Database Redundancy:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-high-availability#availability-zones
Read Scale-Out:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-read-scale-out



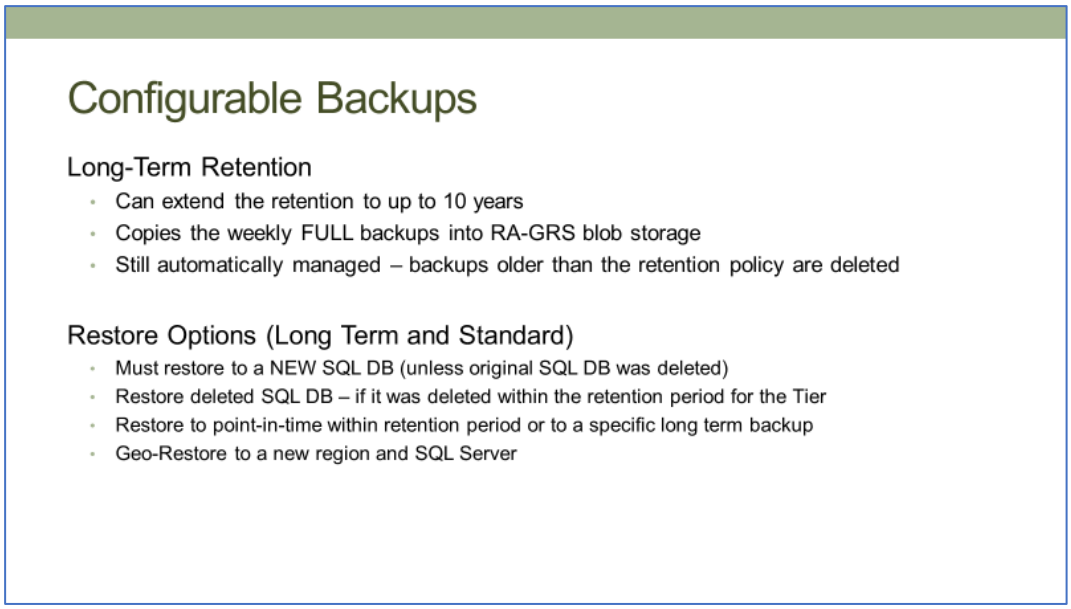
Azure SQL Database takes care of the immediate backup requirements automatically. Microsoft managed processes ensure that you have point-in-time recovery for your databases for up to 35 days (configurable by you). The backups are stored on RA-GRS blob storage which ensures that a copy is stored in the secondary region as well as the primary. However, note that Point-in-Time restores can only be done to the same logical SQL Server as the original database.

You do not have any control over these backups, other than adjusting the retention of the backups between 7 and 35 days.

These backups are also retained for the same duration if you delete a database. For example, if you had your backup retention configured for 7 days then you can still restore it up to the point of deletion until 7 days have passed in which case the backups have been aged out and are no longer accessible. However, if you delete the logical SQL Server, then ALL databases and ALL backups related to those databases are immediately deleted and cannot be recovered.

To ensure the recoverability of these backups, Microsoft periodically tests the restore of automated backups. When the test restores occur a DBCC CHECKDB is also performed to ensure the database integrity. Any issues will be alerted to the Azure SQL Database Engineering Team and communicated to the user.

When restoring a database, it is always restored to a NEW database. This will then allow you to either replace the existing database with the restored one or selectively copy the required data from the restored copy to the original.



Regulatory Compliance may require you to retain backups for considerably longer than 35 days – often up to 10 years. This is addressed through the Long-Term Retention option for backups.

Long-Term Retention

This makes use of the same process that takes the automatic backups (which means that you do not have control at that level). You set the retention policy per Azure SQL Database and the service will ensure that an appropriate backup is copied to long-term storage. The Long-Term Retention (LTR) backup is based on the Full Backup taken on a weekly basis by the automatic process and is managed by the storage service so it has no impact on the database performance.

The retention policy is built from 4 parameters to define the schedule and retention:

- W – defines the weekly retention
- M – defines the monthly retention
- Y – defines the yearly retention
- WeekOfYear – defines which week number is retained for the yearly scope

It is possible to have a combination of all of these defined in the retention policy. For example:

W=0, M=3, Y=5, WeekOfYear=3	This would have the first full backup taken each month to be retained for 3 months and the backup taken during the 3rd week of the year to be retained for 5 years
W=52, M=0, Y=0	The weekly backups are retained for 52 weeks (1 year)
W=6, M=12, Y=10, WeekOfYear=16	The weekly backups are each retained for 6 weeks, except for the first backup each month which is instead retained for 12 months and the backup during the 16th week of the year which is kept for 10 years

To be able to configure LTR using PowerShell, you must have either the Azure PowerShell Az module or AzureRM.Sql-4.5.0 or newer and AzureRM-6.1.0 or newer installed. In addition, there are RBAC restrictions in the subscription to ensure that the user attempting to configure or access the LTR backups is either the Subscription Owner or a member of either SQL Server Contributor or SQL Database Contributor. It is important that this access is carefully controlled, especially if the backups are for compliance reasons. At a more granular level, the individual permissions required are:

To view and restore from LTR:

Microsoft.Sql/locations/longTermRetentionBackups/read
Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionBackups/read
Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/read

To remove the LTR policy:
Microsoft.Sql/locations/longTermRetentionServers/longTermRetentionDatabases/longTermRetentionBackups/delete

The PowerShell cmdlets are:

Create or clear a Policy	Set-AzSqlDatabaseBackupLongTermRetentionPolicy
View Policies	Get-AzSqlDatabaseBackupLongTermRetentionPolicy
View LTR Backups	Get-AzSqlDatabaseLongTermRetentionBackup
Delete LTR Backup	Remove-AzSqlDatabaseLongTermRetentionBackup
Restore from LTR Backup	Restore-AzSqlDatabase –FromLongTermRetentionBackup

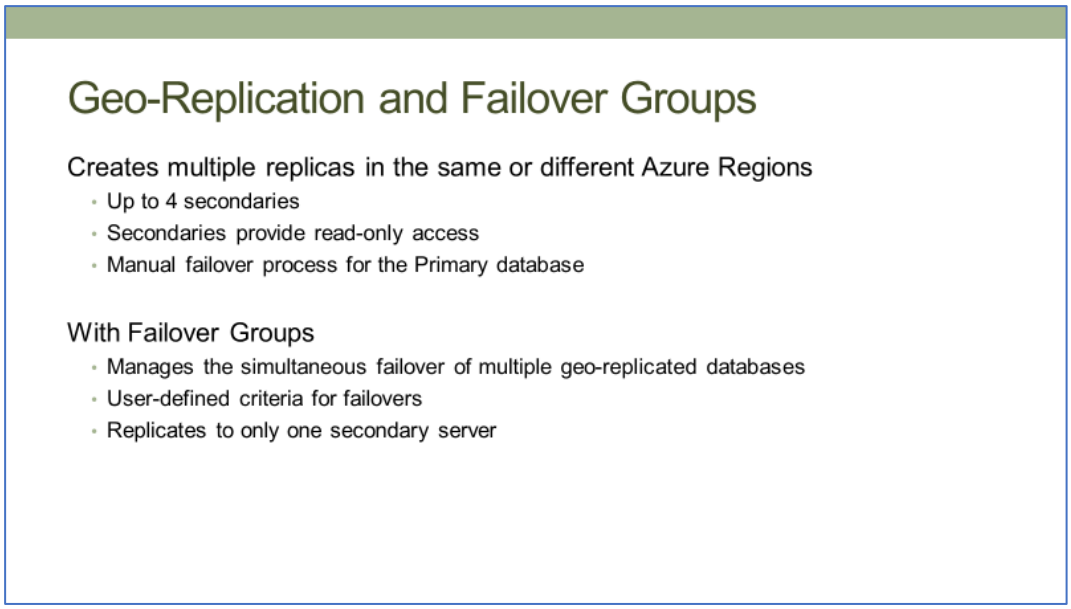
Previously, LTR was configured to use the Azure Services Recovery Vault. This format has been deprecated and access to the API was removed at the end of May 2018.

Restoring a Database

If the original SQL Server is available then a point-in-time restore of a database can be carried out. As the Transaction Logs are backed up every 5 minutes or so, you cannot restore to a time after approx. 6 minutes ago.

If you are restoring to a different logical SQL Server then you do not have the choice to select the point-in-time but will only be able to choose between either the most recent Geo-Restored Backup or one of the backups stored in LTR.

There is a delay in the transfer of the backup files available for Geo-Restore from the primary region to the secondary. This is normally around 1 hour delay therefore there will be an RPO of at least 1 hour when using Geo-Restored backups.



With the default settings, Geo-Restore can only satisfy an RPO of about 1 hour (based on the frequency of the copies to the secondary region). To get a lower RPO, Active Geo-Replication is the feature that will provide readable replicas of your database in the same or a different region.

There are two features that provide similar levels of redundancy and availability.

Active Geo-Replication

This feature provides the following capabilities:

Automatic Asynchronous Replication – the secondary database can be created in any Azure region and logical SQL Server. Once created the database is seeded with data copied from the primary and then continues to receive asynchronous updates from the primary.

Readable Secondary Databases – the secondary replica can be used for read-only operations. The secondary database operates in snapshot isolation mode to ensure that replication of the updates from the primary (log replay) is not blocked and delayed by queries executing against the secondary. Multiple secondaries can be added into the topology. Each primary can support up to 4 secondaries, however each secondary can also act as a primary to replicate to a further 4 secondaries (chaining). This is similar to a Distributed Availability Group configuration in the SQL Server world.

Elastic Pools – each database is independent and can individually be part of an Elastic Pool to support the efficient use of resources in each Azure Region. Since each replica is in a different Azure Region, they will never be members of the same Elastic Pool.

Performance and Service Tiers – each of the databases MUST be in the same Service Tier (Basic, Premium, General Purpose, etc) but do not have to have the same Performance Tier within this. However, there is a risk that if you create the secondary with lower resources then in the event of a failover and that secondary becoming the primary then the application may be impacted. The Performance Tier of any database in a geo-replication configuration cannot be changed unless all partners are online – therefore the secondary will not be scalable until the outage is rectified which defeats the purpose (in part). If you do create the secondary with lower resources then the Log IO Percentage Usage of the primary is the defining metric to consider. If

for example the primary is a P6 (1000 DTU) and the Log IO is 50% then the secondary will need at least 500 DTU available which maps to a P4.

User Controlled Failovers –In an outage scenario, when selecting the new primary the "Unplanned" option should be used as that immediately promotes the secondary to primary. This also remaps any other secondary in the topology to accept replicated transactions from the new primary. Since the process is asynchronous, an Unplanned failover could result in a small amount of data loss if the latest transactions have not yet been sent from the original primary. When the original primary is brought back online, the system automatically reverts it to a secondary and brings it back up to date with the new primary. When doing planned failovers, then the "Planned" option should be used to move things in a more controlled manner with no data loss.

Design Considerations – When using geo-replication, it is recommended that the Firewall Rules are defined at the database level. This means that all databases in the topology have the same set of rules defined and no additional management overhead is needed to replicate rules at the logical SQL Server level. The same reasons suggest using contained database users or Azure AD users/groups.

Connection Strings – since each Azure SQL Database in the topology is useable and may be connected to individually, the application connection string may need to be updated after a failover to point to the new primary.

Auto-Failover Groups

Failover Groups are built upon and extend some of the functionality of geo-replication. They support group level replication and automatic failover and also add the concept of a listener endpoint which can remove the need to update connection strings.

A Failover Group is created between two logical SQL Servers in different Azure Regions. There can be multiple Failover Groups defined between a given pair of SQL Servers and each group can contain one or more Azure SQL Databases that are recovered as a unit if any or all of the primary databases become unavailable.

There are some additional rules around adding Azure SQL Databases to the Failover Group that are additional or different to the base geo-replication:

- Adding a single Azure SQL Database to a Failover Group creates a secondary database with the same Service Tier and Performance Tier to the secondary SQL Server.
- If the primary database is part of an Elastic Pool, then the secondary is created in an Elastic Pool with the same name.
- If the primary database already has a secondary through geo-replication then that geo-replication is inherited by the Failover Group. However, if the secondary is not the same server that is part of the Failover Group then a new secondary is created in the secondary server with the same name.

The Failover Group specific properties are:

Failover Group read-write listener – this creates a DNS CNAME record in the format <failover-group-name>.database.windows.net This will point to the primary server URL and allows transparent reconnect to the new primary in the event of a failover.

Failover Group read-only listener – this is also a DNS CNAME record in the form <failover-group-name>.secondary.database.windows.net that will point to the secondary server URL for read-only connections.

Automatic Failover Policy – there is always a default policy created for a Failover Group which is automatic failover. The failover is trigger if a failure is detected and the grace period has expired. This allows time for the system to check that it is a serious enough outage before performing the failover. You can however turn this off and use manual failover only where you have complete control.

Read-Only Failover Policy – normally the read-only listener cannot be failed over. This ensures that performance on the primary is not impacted if the secondary goes offline and read-only traffic is added to the read-write traffic. If you know you have headroom or you need read-only connections to be maintained then you can enable failover for the read-only listener.

Manual Failover – even with automatic failover, you can still perform a manual failover at any time. Like the geo-replication there is 2 modes – forced or friendly – with the difference being that "friendly" will ensure full synchronisation between the databases before completing the failover and updating the DNS records.

Grace Period with Data Loss – because asynchronous replication is used, a failover has the risk of some data loss. To help mitigate this you can set how long the system will wait from when an outage is detected before it will attempt to perform the failover. This helps protect from transient failures – if the outage is mitigated before the end of the grace period then no failover is undertaken. There are also additional checks which identify that the databases are still online (e.g. the outage has only impacted the service control plane). In this scenario the service will attempt a "friendly" failover to ensure no data loss occurs no matter what value is set in the Grace Period.

Scalability with Failover Groups

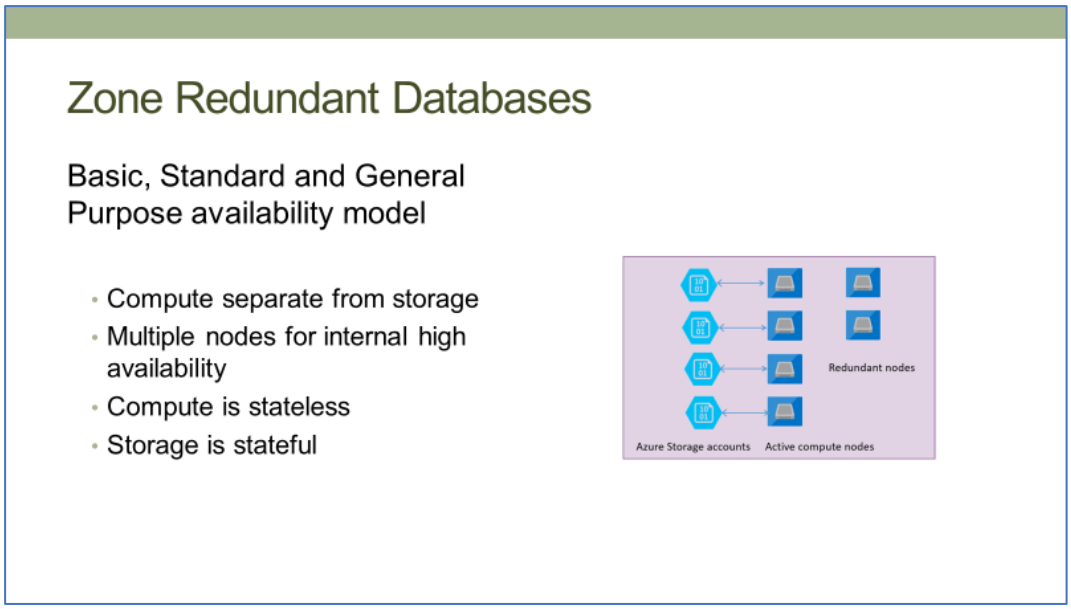
It is possible to vary the compute size within the same service tier without disconnecting the secondary database. When doing this the recommended order is:

- Upgrading: upgrade the secondary first then the primary
- Downgrading: downgrade the primary first then the secondary

Note, however when moving between service tiers the ordering is enforced.

DEMO 7: Tour the Azure Portal and show:

- Create new database from backup (from the SQL Server blade)
- Configure LTR (from Manage Backups on the SQL Server blade)
- Show Failover Groups (from the SQL Server blade)
- Restore to point in time (from database blade)
- Show Geo Replication (from the database blade)



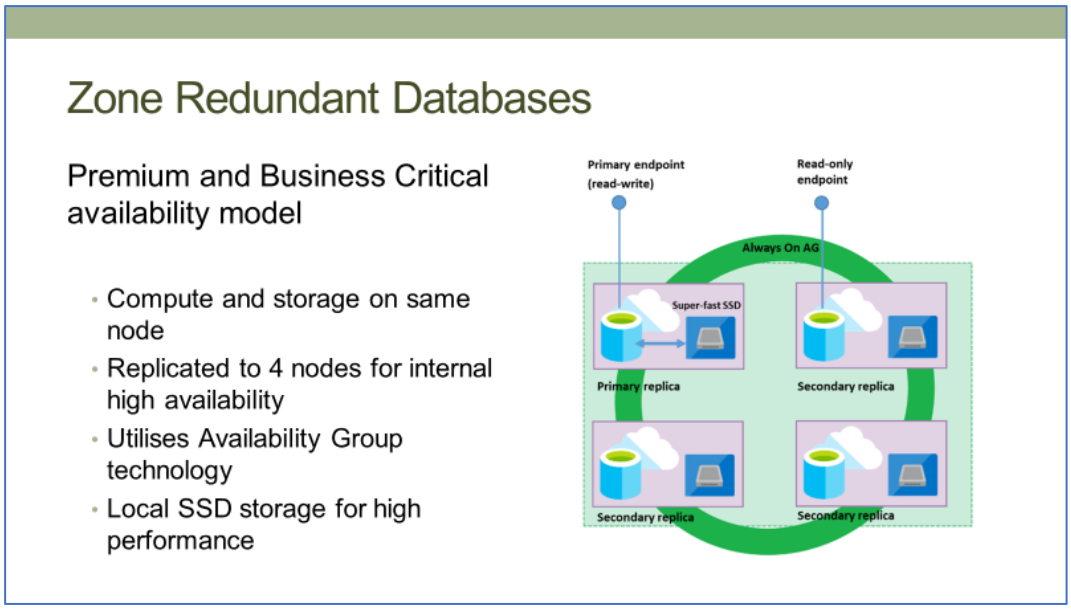
For the Basic, Standard and General Purpose tiers, the 99.99% SLA is achieved through the separation of compute and storage combined with replication of the data in the storage tier.

The compute layer is stateless and runs the sqlserver.exe process. It will hold transient or cached data (e.g., buffer pool). It is managed by the Azure Service Fabric which will perform a failover to another node if required.

The storage layer holds the database mdf and ldf files in Azure Premium Storage. There is a guarantee of no data loss for any record placed in any database file. The redundancy in Azure Storage ensures the data is maintained even if the compute process crashes.

When Microsoft perform patching or if some underlying infrastructure fails, the Service Fabric moves the stateless SQL Server process to a new compute node. There are always some spare nodes waiting to run which minimises failover time – the data and logs are then attached to the new compute node.

This configuration guarantees the availability but could have some performance impacts on a heavy workload due to the transition time and also the initial cold cache.



For Premium and Business Critical tiers, the infrastructure is designed for intensive workloads that cannot tolerate any performance impact. Here compute and storage are co-located to get the benefits of fast SSDs with low latency.

The redundancy is achieved through a 4-node Availability Group cluster.

The primary node constantly pushes changes to the secondary nodes in order and ensures that the data is synchronized to at least one secondary replica before committing each transaction. This process guarantees that if the primary node crashes for any reason, there is always a fully synchronized node to fail over to.

If the primary replica crashes then the standard SQL Server Database Engine AG processes handle the failover to one of the secondary nodes. In addition, a new secondary node is configured and added to the cluster to maintain the 4-node redundancy. Workload is automatically redirected to the new primary node.

Read Scale-Out

With this architecture, all of the replicas are provisioned with the same compute size. With Read Scale-Out (currently in Preview) enabled in the Premium or Business Critical Tiers, you can use one of the replicas for read-only workloads for free – removing this load from the read-write primary replica.

This must be explicitly enabled on the database before trying to use it. If it is not enabled or if the Performance Tier is not Premium or Business Critical then the traffic is directed only to the read-write primary. It can either be enabled at creation time or updated later with PowerShell or REST API.

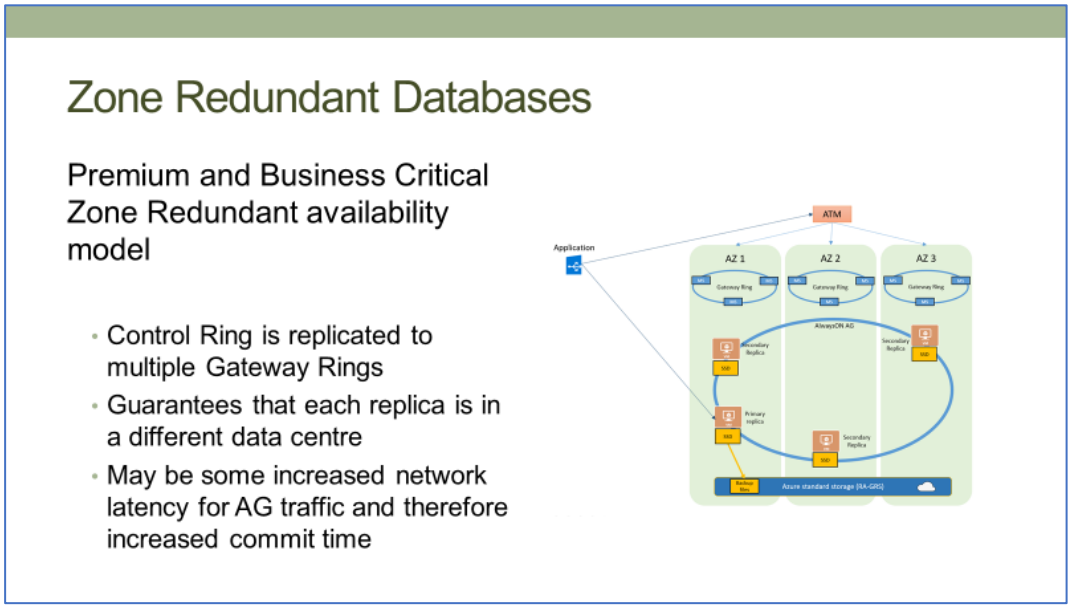
The connection string must then include the *ApplicationIntent* property to be able to be redirected to the read-only replica.

Note however that since the replication is asynchronous, there is a small chance that reading after a write may direct you to a replica where the transaction log has not been fully replayed and your data would appear to be missing.

Read Scale-Out and Geo-Replication

If using Read Scale-Out with a Geo-Replicated Azure SQL Database, then it should be enabled on both the primary and the geo-replicated secondary, this will ensure that in the event of a failover that the connections still behave as expected and that you retain the same load balancing. If you connect to the Geo-Replicated Secondary with Read Scale-Out enabled then be aware of the following behaviours:

ApplicationIntent=ReadOnly	connections will be routed to one of the secondary copies on the Geo-Replicated secondary
Without ApplicationIntent=ReadWrite	connections will be routed to the primary of the Geo-Replicated secondary. Recall that the Geo-Replicated secondary has a different end-point

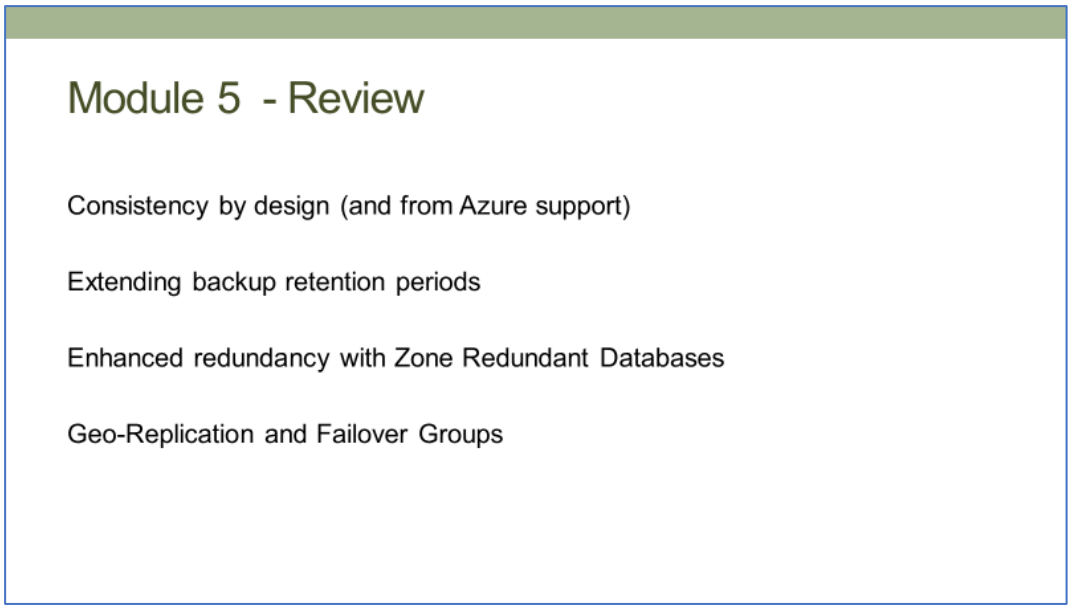


Zone Redundant configuration only applies to the Premium and Business Critical tiers as it is an extension of the Availability Group configuration used there.

In the AG configuration, each of the replicas are created in the same Data Centre. With the introduction of Azure Availability Zones, there is now the ability to have the different replicas in the quorum set placed in different availability zones in the same region.

To eliminate any single point of failure, the Control Ring is also duplicated across multiple zones as 3 Gateway Rings. The Azure Traffic Manager (ATM) controls the routing to a specific gateway ring.

Because the quorum set is now distributed across multiple data centres your database is now more resilient to a greater set of failures. Also, since it does not add any additional database specific redundancy, it is available for you to use with no additional cost.



Summary

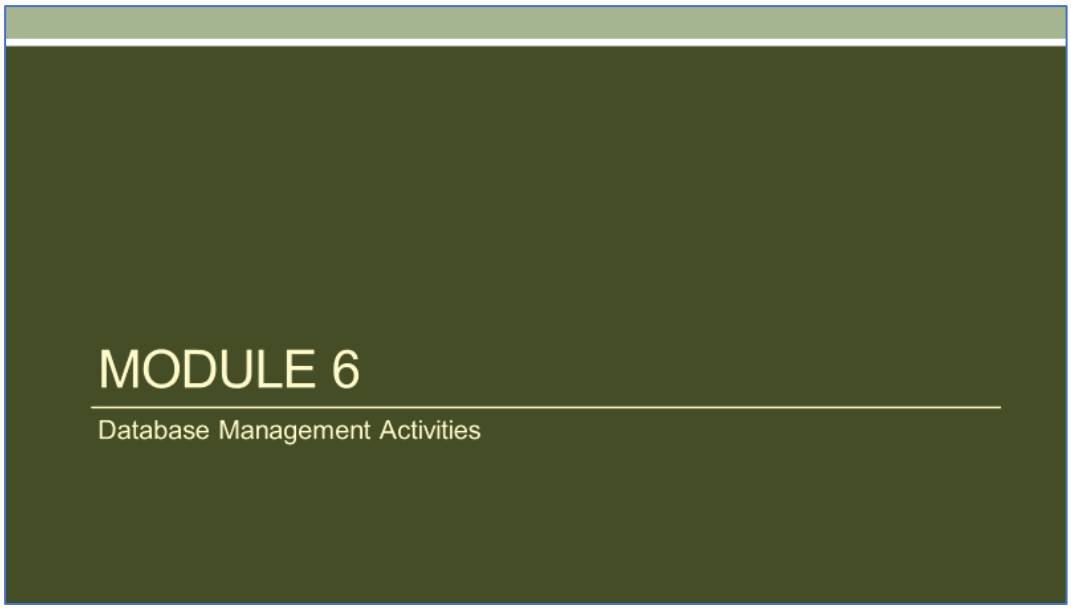
In this module we started covering the additional HA and DR capabilities of Azure SQL Database.

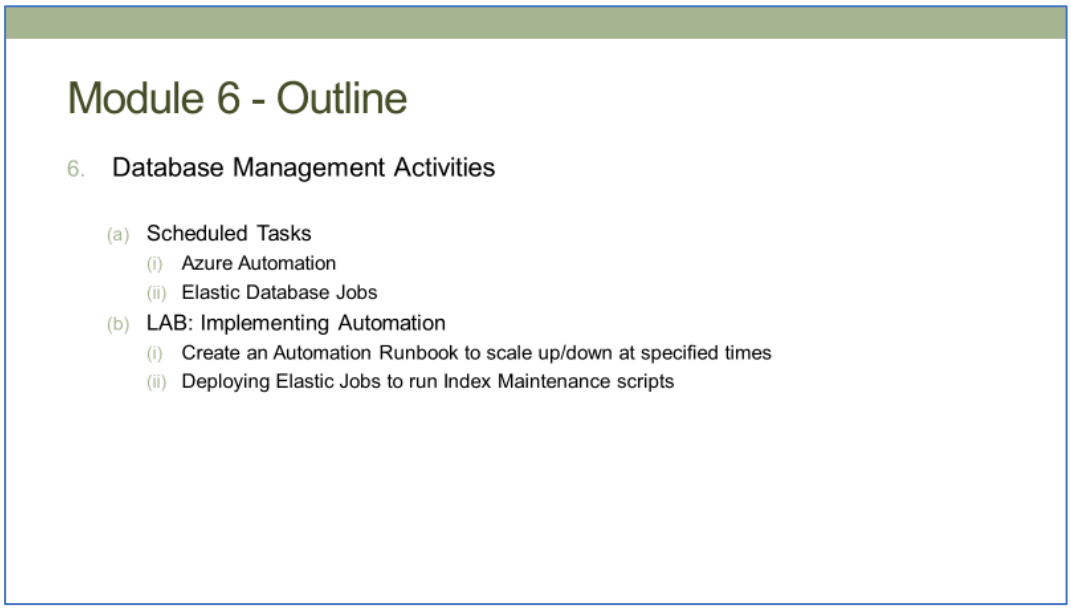
The configuration of the service means that you immediately get a very robust environment with multiple copies kept. However sometimes you need more – whether scaling globally to have data closer to users or just separating read-write from read-only workloads. The physical configuration of the different tiers and also with Zone Redundancy were shown.

The Azure Engineers are very active behind the scenes with checks on your data integrity. This is all included in the service.

Beyond the basics, we looked at:

- Extending backup retention
- Geo Replication at the individual database level
- Failover Groups for multiple sets of databases which are like availability groups. This enables a secondary DNS name for read-only access





Introduction

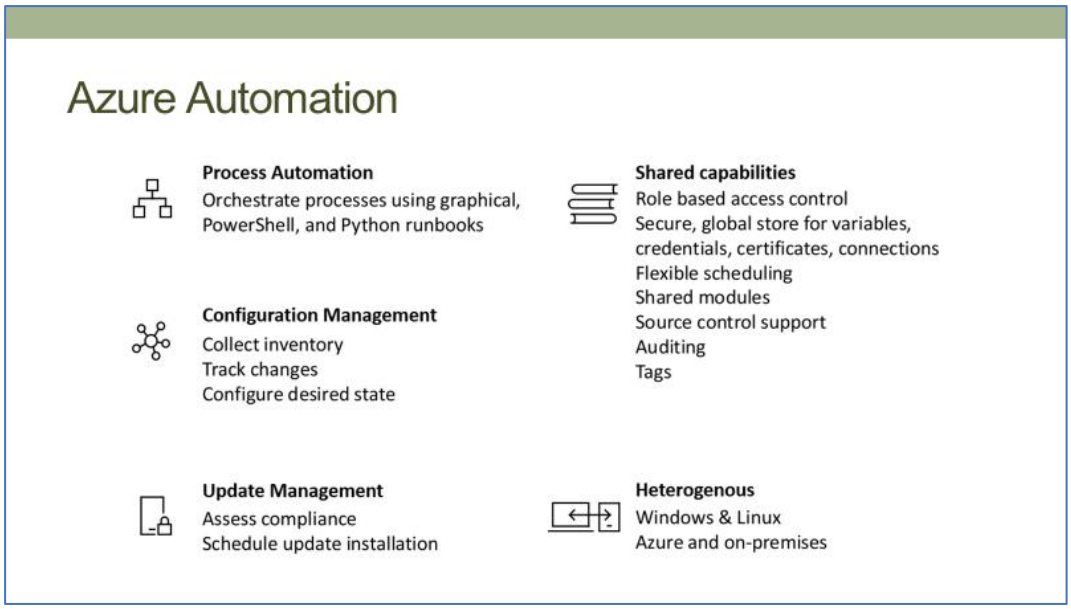
This module will cover the different ways to schedule activities to happen against your Azure SQL Database. As we found out, there is no SQL Agent with Azure SQL Database and therefore we need to consider the other options within Azure.

We will look at the generic Azure Automation – an Azure Service that can interact with any other service in Azure. We will also look at the NEW Elastic Jobs which has recently replaced the customer-hosted Azure Cloud Service version.

There is a LAB at the end of this Module where you will perform some automation tasks.

Resources

Azure Automation Overview:	https://docs.microsoft.com/en-us/azure/automation/automation-intro
Creating Azure Automation Runbooks:	https://docs.microsoft.com/en-us/azure/automation/automation-creating-importing-runbook
Azure Automation for Azure SQL Database:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-manage-automation
Elastic Jobs (Deprecated Cloud Service):	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-elastic-jobs-overview
Elastic Database Jobs:	https://docs.microsoft.com/en-us/azure/sql-database/elastic-jobs-overview
Learning PowerShell Workflow:	https://docs.microsoft.com/en-us/azure/automation/automation-powershell-workflow
Azure SQL Database PowerShell Reference:	https://docs.microsoft.com/en-us/powershell/module/servicemanagement/azure/?view=azuresmps-4.0.0#sql
Run TSQL Commands in Azure Runbook:	https://gallery.technet.microsoft.com/scriptcenter/How-to-use-a-SQL-Command-be77f9d2
Elastic Database Jobs Procedures:	https://docs.microsoft.com/en-us/azure/sql-database/elastic-jobs-tsql#job-stored-procedures
Elastic Database Jobs PowerShell Reference:	https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/sql-database/elastic-jobs-powershell.md



Azure Automation

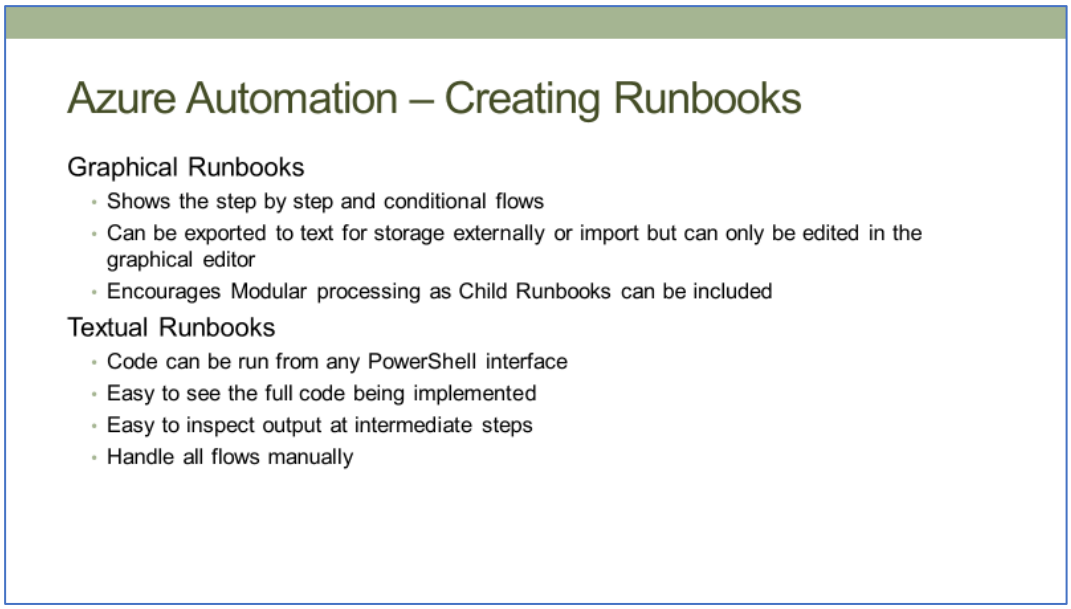
Azure Automation is an Azure wide service for automating anything that can be done using the many Azure PowerShell modules or Python scripts.

Depending upon what you want to do, you may have to import the specific PowerShell module or Python module into your automation environment – this may be more important if new functionality is added to the modules as this is not always released into Automation at the same time.

The PowerShell options encompass both scripts and PowerShell Workflow. A PowerShell Workflow adds the ability to simultaneously perform an action against multiple devices and the ability to recover from failure using checkpoints.

You create a new runbook either by creating it in the Azure Portal, or by importing from a file or the Runbook Gallery. Before this you need to have created your Automation Account. When you create it you should enable the *Run As Account* – this will create a new service principal user in Azure AD with the Contributor Role in the Subscription, effectively sufficient permissions to act across all your resources as required. If you need to set more granular permissions, you can remove it from the Contributor Role in the Subscription and instead add it to e.g. the Contributor Role for a specific Resource Group.

Note that Automation accounts are not available in ALL Azure Regions, but do appear to be in each global area. For Australia it is only in the Australia Southeast region.



There are a number of methods to create an Automation Runbook and different types of coding styles that can be used. These can be summarised as:

- PowerShell — Either graphical or textual
- PowerShell Workflow — Either graphical or textual
- Python — Textual only

Graphical Runbooks

The graphical option provides a GUI based interface with a drag-and-drop type style interface to add items such as individual PowerShell cmdlets. It also easily exposes shared assets such as the Run As credential and other Runbooks in the account which can be called as Child Runbooks. The flows between tasks can be made conditional on outputs from a previous step and can also be configured to have alternative conditional flows or even junctions which rely on all input paths having completed successfully.

These help to visualise what is happening within the runbook process.

The runbook can be based on PowerShell script, or PowerShell Workflows depending upon the requirements for parallel processing and restart ability.

Textual Runbooks

These have the flexibility of producing the code through any suitable editor and imported into the Automation Account. It is currently the only choice if using Python as the coding language.

Complex logic may be easier to implement in a textual runbook as you have full access to the code. All process logic between steps has to be coded in your PowerShell / Python scripts.

Runbook Development

There is a core set of PowerShell Modules and Python Packages that are available by default. Additional modules and packages can be imported into your Automation Account.

Whether you decide to build your runbook using Graphical or Textual methods, the initial development flow and script requirements are much the same.

1. Configure any Global Variables that you may require. The SubscriptionID is a useful one to ensure is defined.

2. In the Azure Portal create a new Runbook of the appropriate type. The items that are available for selection are the same for Cmdlets, Assets and other Runbooks that need to be referenced. However Graphical Runbooks add additional functionality for Runbook Control which can add Junctions or other code items that are too complex for the standard Cmdlets to achieve.

3. Make sure the first steps in your Runbooks are to:

    1. Establish the RunAs Account as the connection mechanism. In graphical this simply adds the appropriate step, in textual it adds a basic command that you should update to store the result and use the returned properties in the next steps.

    2. Login to Azure. Using graphical you would select the Connect-AzAccount Cmdlet and use the *ServicePrincipalCertificate* parameter set and in textual simply use *Connect-AzAccount* with appropriate parameters.

3. Set the correct context so that the remaining activities are connected in the right place. For graphical, use the Set-AzContext Cmdlet with the *SubscriptionId* parameter set and for textual simply use *Select-AzContext* with the appropriate parameters to return a context that can be used for the later Cmdlets.

4. Add parameters. For the graphical option, identify the parameters by clicking on *Input and Output* at the top of the pane and add what you need. For textual, these are defined in the same way as any other PowerShell script parameters at the start of the script using:

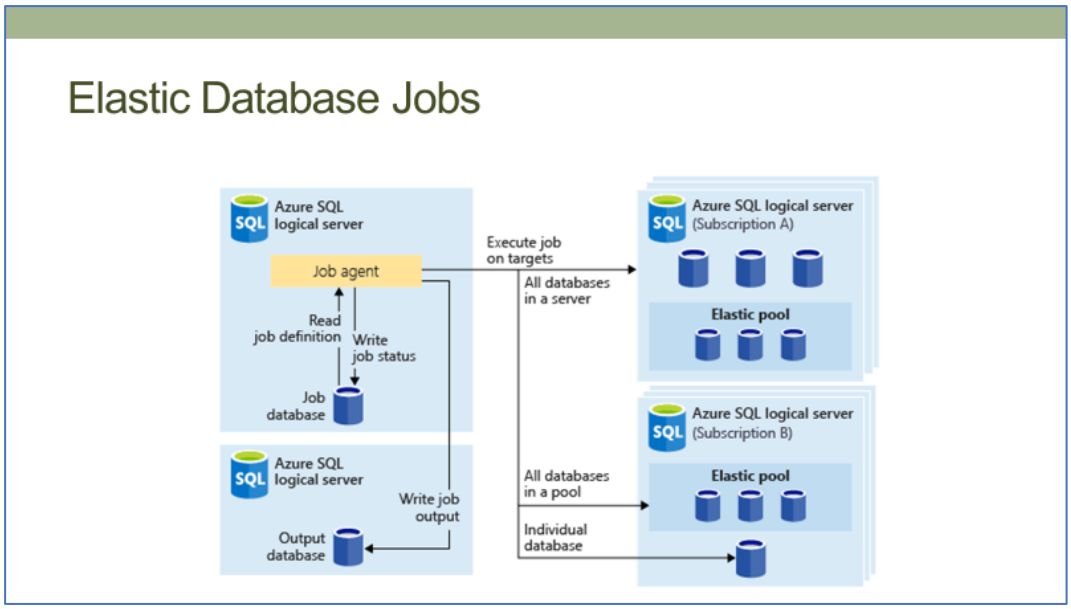
```
Param(
        [string]$VMName,
        [string]$ResourceGroupName
)
```

In the graphical runbook development environment, if you want to use the parameters in any specific step then there is the section on the blade to access and define the parameter mappings.

When developing in the Azure Portal (graphical or textual) there is a Test button on the blade where you can configure any input parameters and the level of logging and then run a test execution. Once you have completed debugging then you publish the Runbook and make it available to be executed on a schedule or on demand and as we will see in Module 7, we can also have it executing in response to an alert from Azure.

DEMO 8: Show the graphical capabilities for developing a runbook:

- In SharedResources in the AutomationAccount, add a variable for SubscriptionId and enter your SubscriptionId string
- Create a new Runbook
- Add Assets\Connections\AzureRunAsConnection
- Add Cmdlet\Add-AzureRmAccount
    - Select the ServicePrincipalCertificate parameter set
    - Configure TenantId, ApplicationId and CertificateThumbprint from ActivityOutput of the first task. Set ServicePrincipal to be a constant value of TRUE
- Add Cmdlet\Set-AzureRmContext
    - Select the SubscriptionId parameter set
    - Set the SubscriptionId parameter to the variable
- Add Cmdlet\Get-AzureRmSqlDatabase
    - Set the parameter values manually as constants – only use the server name, not the full path
- Add Cmdlet\Set-AzureRmSqlDatabase
    - Set ResourceGroupName, DatabaseName and ServerName as constant values
    - Set RequestedServiceObjectiveName to S0 (Current setting should be Basic)
- Update the linkage between the last 2 tasks to Sequence and ApplyCondition
    - Set condition to $ActivityOutput['Get-AzureRmSqlDatabase'].CurrentServiceObjectiveName -eq "Basic“
- Click on the Test Pane to run the Runbook and show the DB has scaled up



The original release of Elastic Jobs required the deployment of an Azure Cloud Service within your tenancy. This Cloud Service was then able to make use of the elastic database libraries. The service also required the creation of a control database which stored all meta-data and state data for the jobs and two worker roles to act as the controller to coordinate the execution of the jobs and the execution of the actual job tasks.

The jobs were able to be targeted to either a Shard Map group, a single Azure SQL Database or a custom collection of Azure SQL Databases.

This has now been deprecated and replaced by an Azure Hosted Elastic Database Job engine which sits far more like a SQL Agent in the cloud.

New Elastic Database Jobs

Elastic Database Jobs provide the ability to run one or more T-SQL scripts in parallel, across a large number of Azure SQL Databases, on a schedule or on-demand.

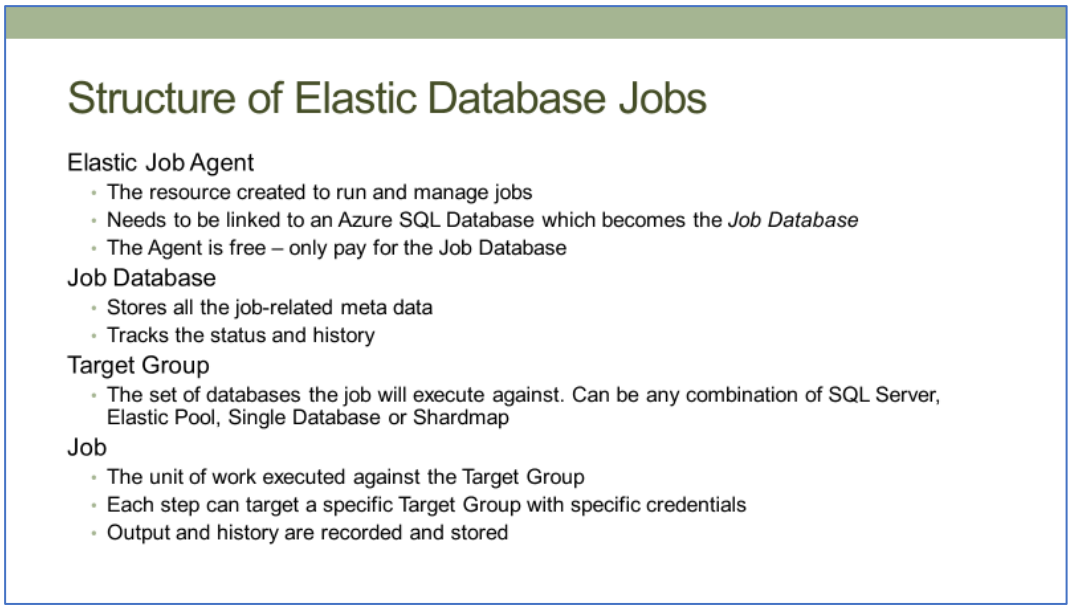
Jobs can be executed against any combination of databases: one or more individual Azure SQL Database, all Azure SQL Databases on a logical SQL Server, all Azure SQL Databases in an elastic pool, or shardmap, with the added flexibility to include or exclude any specific database.

Jobs can run across multiple logical SQL Servers, multiple elastic pools, and can even run against Azure SQL Databases in different subscriptions. Servers and pools are dynamically enumerated at runtime, so jobs run against all databases that exist in the target group at the time of execution.

The new model provides a number of benefits:

Manage many databases at the same time	Schedule administrative tasks to run on a recurring schedule. Run common scripts to e.g. deploy schema changes or collect performance data common across all databases. Collect results from a query executed against a set of databases into a central table on an on-going

	basis, allowing performance queries to be continually executed.
Collect data for reporting	Aggregate data from a collection of Azure SQL databases into a single destination table.
	Execute longer running data processing queries across a large set of databases, for example the collection of customer telemetry. Results are collected into a single destination table for further analysis.
Reduce overhead	Let the job handle the task of logging in to each database in the target group.
Accounting	Jobs log the status of execution for each database.
	You also get automatic retry when failures occur.
Flexibility	Define custom groups of Azure SQL databases, and custom schedules for running the jobs.



Elastic Job Agent

Deploy Elastic Database Jobs created an Azure Resource Group to enable the creation, execution and management of the Elastic Jobs. It requires an empty Azure SQL Database at the S0 Performance Tier (or higher). The tier depends on the number of job steps and how frequently they are executed – so you may find that you need a higher performance tier to reliably run the required jobs.

When the Elastic Job Agent is deployed it creates a schema, tables and a role in the empty database. The *jobs_reader* Role that it creates can be used to allow lower privileged users to monitor job executions without allowing them access to create or edit a job which may expose stored credentials.

Job Structure

For each job step defined, there is a Target Group which identifies which databases the step will be executed against. The Target Group is evaluated at runtime, therefore any changes to e.g. the Azure SQL Databases that are members of a specific Elastic Pool are picked up automatically at each execution. It is also possible to have a logical SQL Server or Elastic Pool as the broad Target Group but exclude specifically named individual Azure SQL Databases.

For example, you may have included your Job Database in an Elastic Pool as it only runs jobs overnight when other databases are inactive. It may be that you do not want to execute against the Job Database specifically so you could target the Elastic Pool with a specific exclusion for the Job Database. The flexibility is also demonstrated by the ability to execute across multiple subscriptions as well as multiple SQL Servers.

Each Job Step is a T-SQL script to be executed against its Target Group. It also contains the credentials that the Job Agent will need to connect to the target database. The timeout and retry policies are defined at the Job Step and it can also specify output parameters.

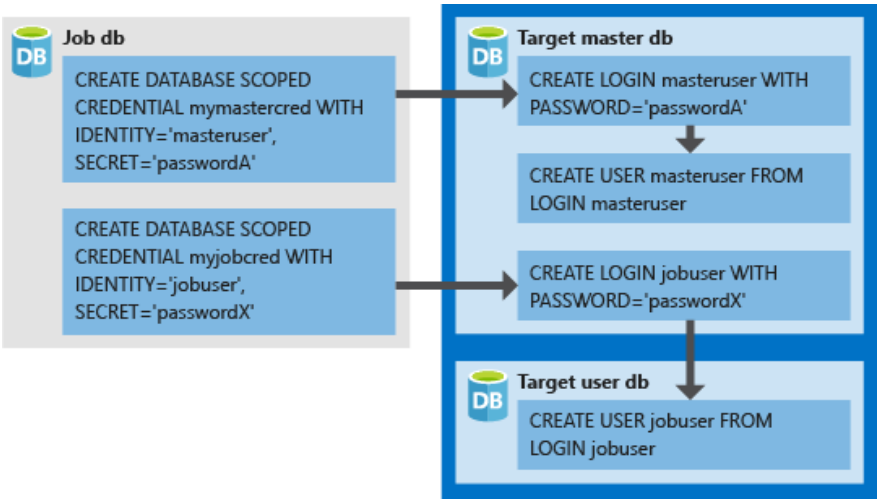
The critical aspect to remember about the Job Step is that the T-SQL script ***must be idempotent***. With the chance of an individual execution against a specific Azure SQL Database failing and the Job Step being retried, you may not want the other successful executions to be retried as this could throw further errors.

The output of the execution against each Azure SQL Database in the Target Group is recorded and can be saved to a table. The output table does not have to be in the Job Database – it can be anywhere that the Job Agent can write to with appropriate credentials.

Job Execution history is logged in the Job Database and has a retention period of 45 days. Similarly, to SQL Agent, there is a stored procedure named *sp_purge_history* that can be used to remove history less than 45 days old.

Job Credentials

Elastic Database Jobs make use of Database Scoped Credentials to establish the connections to the Azure SQL Databases in the Target Group. If a Target Group contains logical SQL Servers or Elastic Pools, then the database scoped credentials are used to connect to the appropriate master database to enumerate the corresponding in the SQL Server/Pool to actually execute against.



The credentials are configured as follows:

- The credentials must be created in the Job Database
- Each target database must have a login which has sufficient permissions to execute the commands in the Job Step
- A separate credential should be created with access only to enumerate the list of databases in the logical SQL Server or Elastic Pools (this is all stored in the master database)

NOTE: the identity has to use SQL Authentication.

Elastic Database Jobs Performance

An Elastic Job will use minimal resources while it is waiting for long-running Job Steps to complete. At this stage, the Job Agent is limited to 100 concurrent jobs. The Job can also be configured to execute against a maximum number of Azure SQL Databases concurrently (even if there are more than this number in the Target Group) – this helps to keep the resources well managed.

DEMO 9: Show the PowerShell method to add the Agent and credentials

Show the TSQL examples to configure a TargetGroup

Summary

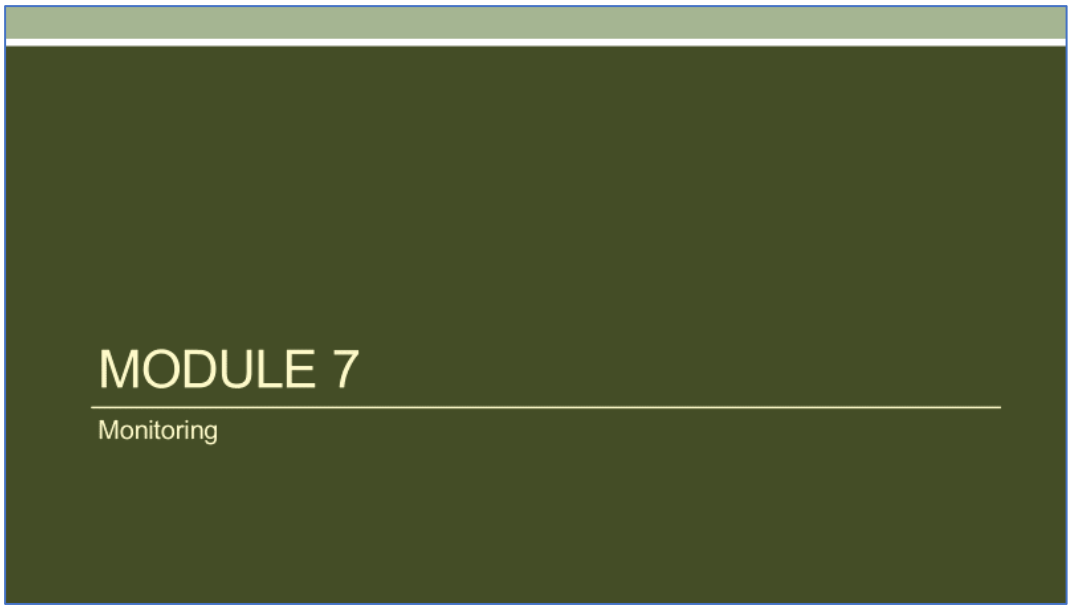
In this module we discovered how we can move away from manual activities and have processes and scripts running automatically for us in Azure in general or specifically in Azure SQL Database.

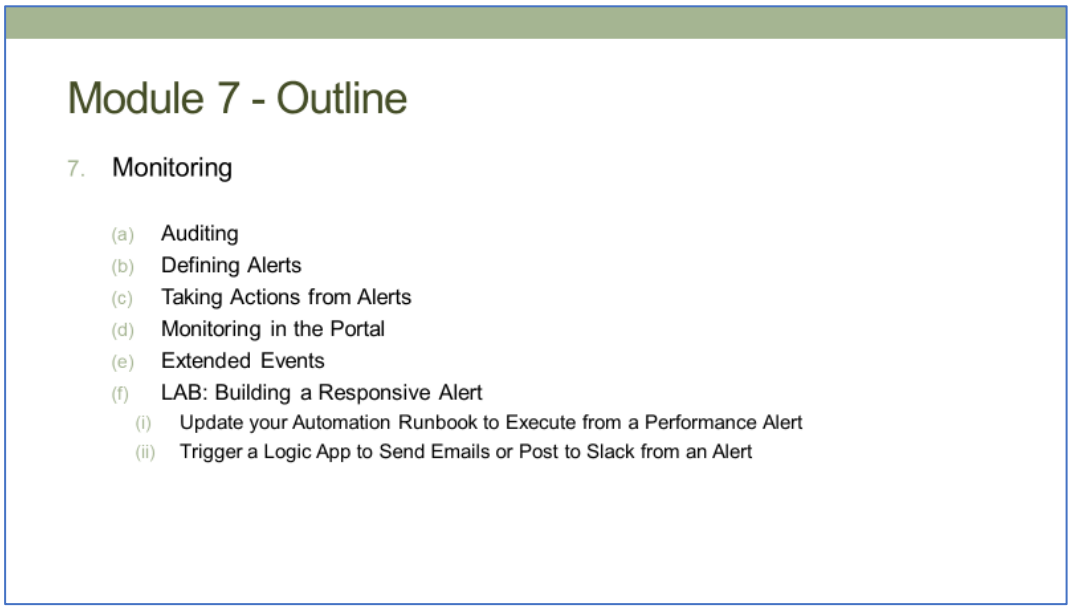
Azure Automation is a powerful engine that can allow us to do almost anything in Azure. It is a generic service that can run PowerShell or Python scripts and touch all of the other services.

We've learnt that Azure Automation Runbooks can be started manually, on a schedule, or as a response to another activity. We can reuse other Runbooks within a Runbook – encouraging modular design.

We also looked at the Azure SQL Database specific Elastic Database Jobs. From the syntax of the job creation and execution statements we could see that it was very similar to the stored procedures related to SQL Agent jobs that live in MSDB. As we need an extra Azure SQL Database created to be the job database, maybe it becomes useful to name the job database "MSDB"? Rather than per server though, we only need to have one for all of our Azure SQL Databases if it is powerful enough.

Elastic Database Jobs are still a Preview feature and need some careful steps at installation time, especially using PowerShell.





Introduction

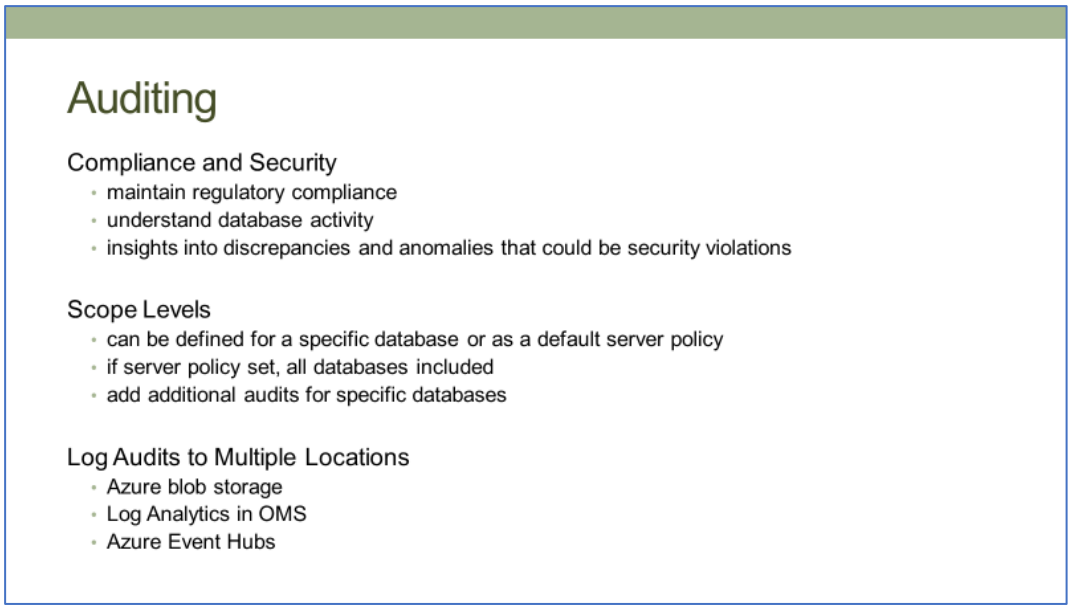
This module will review the different ways that we can work out what is going on under the covers and inside of our Azure SQL Database.

We may need to keep track of who is doing what or just simply save ourselves some time and effort by letting Azure make changes for us that we have allowed it to do.

There is a LAB at the end of this Module where you will update your previously created automation tasks to run as a result of an Alert trigger and also link it further through the Azure ecosystem using Logic Apps.

Resources

Get Started with Auditing:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-auditing
Set Audit Events using PowerShell:	https://docs.microsoft.com/en-us/powershell/module/azurerm.sql/set-azurermsqlserverauditing?view=azurermps-6.10.0
Azure Event Hubs:	https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-about
OMS:	https://docs.microsoft.com/en-us/azure/azure-monitor/overview
Define Alerts in the Azure Portal:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-insights-alerts-portal
Classic Alert Deprecation:	https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-classic-retirement
Unified Alerts:	https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-overview-unified-alerts
Supported Metrics for New Alerts:	https://docs.microsoft.com/en-us/azure/monitoring-and-diagnostics/monitoring-near-real-time-metric-alerts?toc=/azure/azure-monitor/toc.json
Start a Runbook from a Webhook:	https://docs.microsoft.com/en-us/azure/automation/automation-webhooks
LogicApps Quickstart	https://docs.microsoft.com/en-us/azure/logic-apps/
Extended Events for SQL DB:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-xevent-db-diff-from-svr
Common Alert Schema:	https://docs.microsoft.com/en-us/azure/azure-monitor/platform/alerts-common-schema



Why Audit?

There are multiple reasons why you might want to audit the activity on your Azure SQL Database or SQL Server. In some industries, it is defined by regulatory agencies to demonstrate your compliance. Other times it may be an internal process to understand what activity is going on in the database (e.g. confirm whether a table is being used before dropping it). It can also be a rich source of information that can be further analysed to detect trends that may indicate anomalies or suspicious activity.

In Azure SQL Database, we can configure auditing at either the SQL Server level – in which case it is enabled for all databases in that SQL Server. Alternatively, we can configure it at the individual Azure SQL Database level, or in some cases at both. Note that if it is enabled at both levels then there is not consolidation of the audits but they exist side by side. Because of this you should not enable audits at both scopes unless you want to set a different retention policy for a database or audit some additional events.

Within Azure SQL Database, auditing allows you to:

Retain an audit trail of selected events. You can define categories of database actions to be audited.

Report on database activity. You can use pre-configured reports and a dashboard to get started quickly with activity and event reporting.

Analyze reports. You can find suspicious events, unusual activity, and trends.

Audit Destinations

Audits can be written to a choice of destinations. You can choose any one or any combination of the following:

- Azure Storage When you just need to log in a XEL format with a defined retention period
- Log Analytics Writes into OMS which provides a rich query engine for further analysis
- Event Hub Trigger additional actions on the log data as it is written

Audit Actions

The default audit actions created for either a server level or database level audit includes:

BATCH_COMPLETED_GROUP
SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP
FAILED_DATABASE_AUTHENTICATION_GROUP

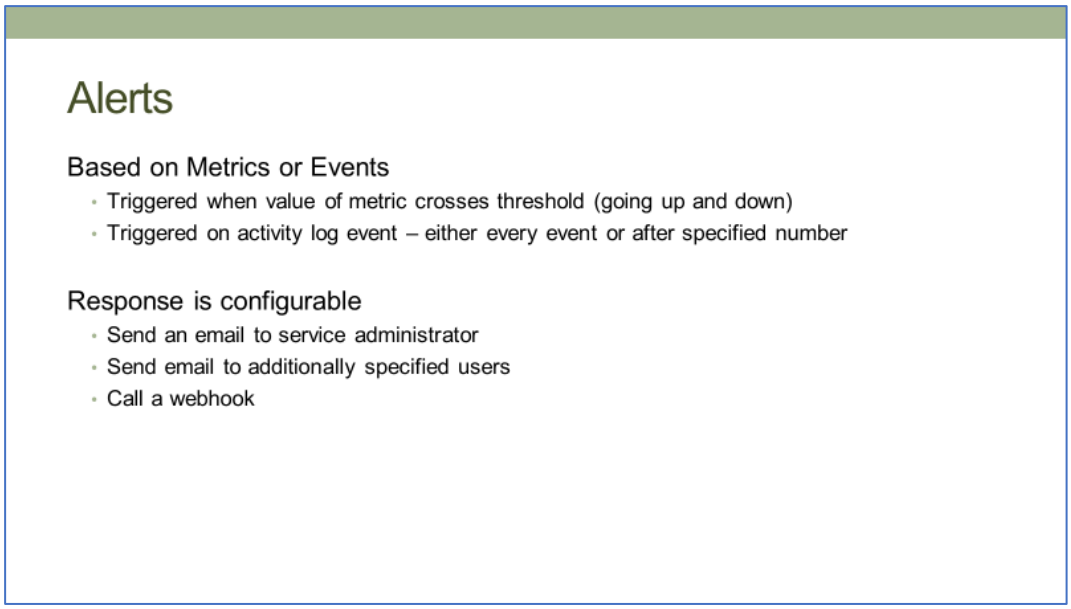
You can update the collection policy to have different events using the PowerShell Cmdlet *Set-AzSqlDatabaseAuditing* or *Set-AzSqlServerAuditing*. The full list of Audit Actions is:

BATCH_STARTED_GROUP,
BATCH_COMPLETED_GROUP,
APPLICATION_ROLE_CHANGE_PASSWORD_GROUP,
BACKUP_RESTORE_GROUP,
DATABASE_LOGOUT_GROUP,
DATABASE_OBJECT_CHANGE_GROUP,
DATABASE_OBJECT_OWNERSHIP_CHANGE_GROUP,
DATABASE_OBJECT_PERMISSION_CHANGE_GROUP,
DATABASE_OPERATION_GROUP,
DATABASE_PERMISSION_CHANGE_GROUP,
DATABASE_PRINCIPAL_CHANGE_GROUP,
DATABASE_PRINCIPAL_IMPERSONATION_GROUP,
DATABASE_ROLE_MEMBER_CHANGE_GROUP,
FAILED_DATABASE_AUTHENTICATION_GROUP,
SCHEMA_OBJECT_ACCESS_GROUP,
SCHEMA_OBJECT_CHANGE_GROUP,
SCHEMA_OBJECT_OWNERSHIP_CHANGE_GROUP,
SCHEMA_OBJECT_PERMISSION_CHANGE_GROUP,
SUCCESSFUL_DATABASE_AUTHENTICATION_GROUP,
USER_CHANGE_PASSWORD_GROUP

Auditing Geo-Replicated Databases

With geo-replicated databases, when you enable auditing on the primary database the secondary database will have an identical auditing policy. If we need this to be different then we would have to have enabled auditing at the server level – in which case each server could have different auditing selections.

Another reason for using server level auditing when geo-replication is involved is that with database-level auditing, the storage settings for the secondary database will be identical to those of the primary database, causing cross-regional traffic. With server level, these are defined separately.



Classic Alerts

Alerting for Azure SQL Database is based on either a metric value crossing a threshold or an event logged in the Activity Log. The Activity Log tracks actions that have occurred in the service at a general level - e.g. an Azure SQL Database was created in a Subscription. These can be picked up and forwarded as an Action to one of the following:

- Email/SMS/Push/Voice
- Azure Function
- LogicApp
- Webhook
- ITSM
- Automation Runbook

A Metric Alert can use a number of counters such as DTU percentage used, can look at averages over a set time range and then notify anyone with the Owner, Contributor or Reader RBAC role membership for that service. Additionally, other users can be explicitly targeted for emails and a Webhook and/or a LogicApp may be triggered.

The Classic Alerts that were used with Azure SQL Database have only recently been identified for retirement. These will be removed from Azure during 2019. The replacement is the unified alerting and monitoring experience from the Azure Monitor service.

Azure Monitor Service

The Azure Monitor Service now provides a single pane of glass type approach to implementing alerting and monitoring across all of the Azure services. Alerts can now trigger Voice Calls, Azure Functions, Logic Apps, SMS, and ITSM Tools like ServiceNow and Automation Runbooks with near real-time monitoring and alerting.

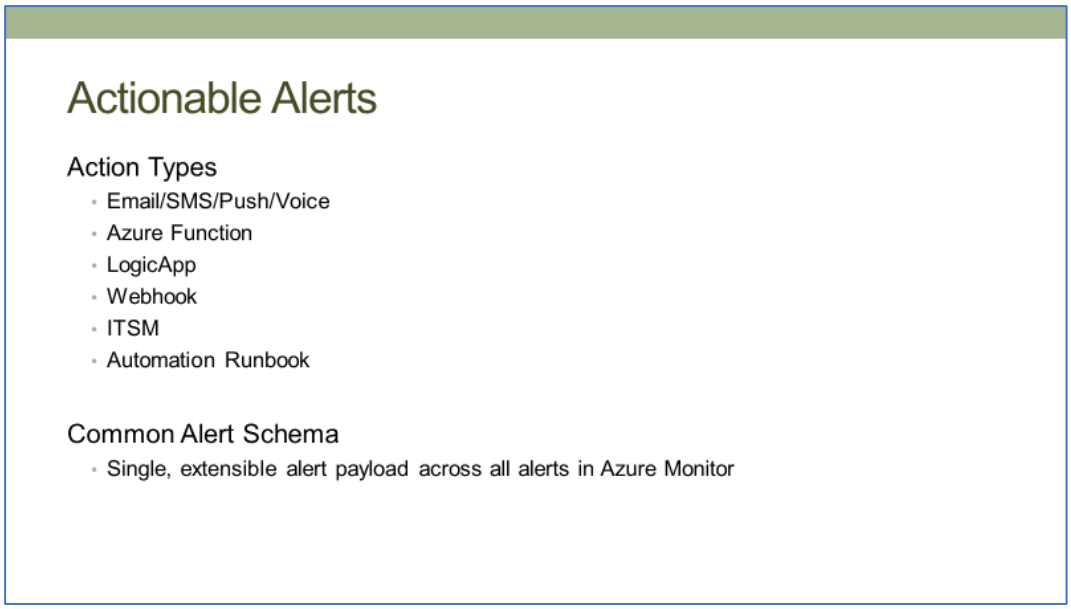
Features of the unified alert experience

The unified experience has the following benefits over the classic experience:

- Better notification system: Action groups are named groups of notifications and actions that can be reused in multiple alerts.
- Unified authoring experience: Alerts and alert rules for metrics, logs, and activity logs across Azure Monitor, Log Analytics, and Application Insights can be managed in one place.
- View fired Log Analytics alerts in the Azure portal: Log Analytics alerts can now be viewed with alerts from other sources in the Azure portal. Previously, alerts from other sources were in a separate portal.
- Separation of fired alerts and alert rules: Alert rules are now distinguished from alerts. An alert rule is the definition of a condition that triggers an alert. An alert is an instance of an alert rule firing.
- Better workflow: The unified alert authoring experience guides you through the process of configuring an alert rule.

Metric alerts have the following improvements over classic metric alerts:

- Improved latency: Metric alerts can run as frequently as once every minute. Classic metric alerts always run at a frequency of once every 5 minutes. Log alerts still have a delay longer than a minute due to the time it takes to ingest the logs.
- Support for multi-dimensional metrics: You can alert on dimensional metrics, which means you can monitor a specific instance of the metric.
- More control over metric conditions: You can define richer alert rules that support monitoring the maximum, minimum, average, and total values of metrics.
- Combined monitoring of multiple metrics: You can monitor up to two metrics with a single rule. An alert is triggered if both metrics breach their respective thresholds for the specified time period.
- Metrics from logs: Log data that's going into Log Analytics can be queried and then alerted on just like other metrics.



We can initiate some further actions in response to an Alert. Currently this exposes Azure Functions, Webhooks, LogicApps, ITSM actions or an Automation Runbook.

Emails/SMS/Push/Voice

Emails will be sent from one of the following email addresses – ensure that you whitelist these:

azure-noreply@microsoft.com
azureemail-noreply@microsoft.com
alerts-noreply@mail.windowsazure.com

These actions are also all Rate Limited in Azure which means that notifications can be suspended when too many are sent to a particular phone number, email address or device. The Rate Limiting Thresholds are:

SMS: No more than 1 SMS every 5 minutes.
Voice: No more than 1 Voice call every 5 minutes.
Email: No more than 100 emails in an hour.

ITSM Connections

These require the IT Service Management Connector to connect Azure to a supported ITSM Product/Service. The connector is bi-directional between Azure and the ITSM tools and currently supports:

ServiceNow
System Center Service Manager
Provance
Cherwell

With the IT Service Management Connector, you can create work items in your ITSM tool based on your Azure alerts (metric alerts, Activity Log alerts and Log Analytics alerts). Optionally, you can also sync your incident and change request data from your ITSM tool to an Azure Log Analytics workspace.

LogicApps

LogicApps simplify how you build automated scalable workflows that integrate apps and data across cloud services and on-premises systems. The service is like having BizTalk in the cloud.

The LogicApp Connectors are the components that let us hook in to various services and applications, pass in our specific payload and then receive output from that service or application which can be used in our next step.

There are many built-in actions and triggers to help you create logic apps that run on custom schedules, communicate with other endpoints, receive and respond to requests, and call Azure functions, Azure API Apps (Web Apps), your own APIs managed and published with Azure API Management, and nested logic apps that can receive requests. You can also use built-in actions that help you organize and control your logic app's workflow, and also work with data.

There are also Managed Connectors which provide triggers and actions for accessing other external services and systems. Some connectors require that you first create connections that are managed by Azure Logic Apps.

Using LogicApps, we can again take the output from our Alert and feed it into the start of a LogicApp workflow which can then perform additional actions for us.

Azure Functions/Webhooks

This allows any piece of code to be executed on demand on a serverless compute service without needing any provisioned infrastructure.

A Webhooks allows an http message to be sent to trigger some further action elsewhere.

Automation Runbooks

The Runbooks that we have developed could also be triggered from an Alert – for example the sample we created to scale a database could be triggered in response to consistently high utilisation on the current performance tier to scale up the resoirces to handle the increased load.

Common Alert Schema

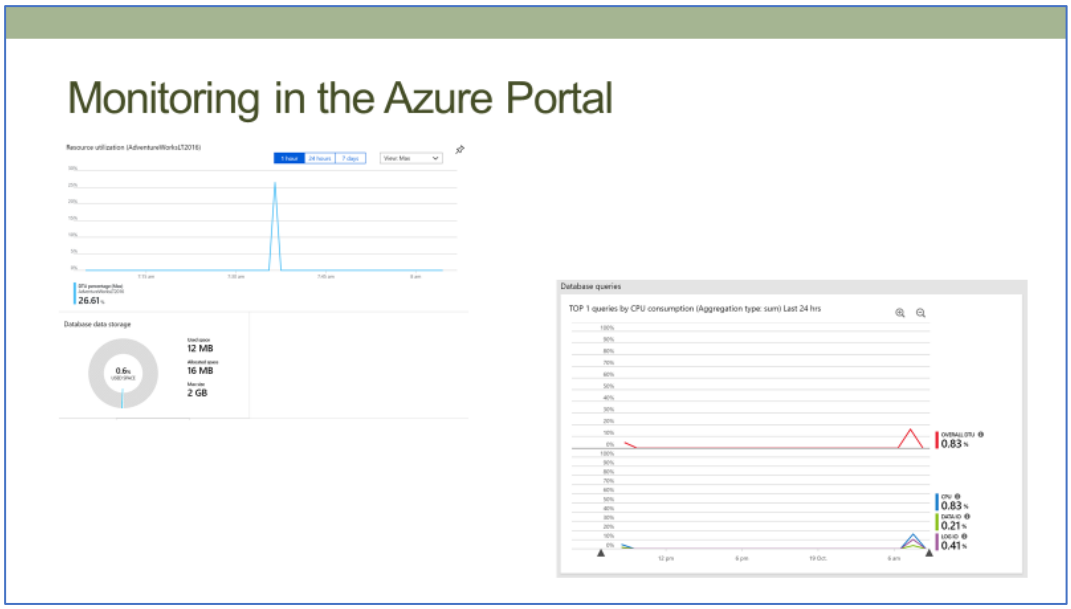
Any alert instance describes the resource that was affected and the cause of the alert. This normally manifests itself in a different JSON schema as the payload of the message for different alert types. The schema details can be found at https://docs.microsoft.com/en-us/azure/azure-monitor/platform/activity-log-alerts-webhook#payload-schema

The Common Alert Schema standardizes the format into two sections:

Essentials A set of fields which are common across all alert types which describe WHAT resource the alert was triggered on along with some common metadata such as severity or description.

Alert Context A set of fields which describe the CAUSE of the alert. These fields vary by alert type. For example a Metric Alert would include the value that triggered it whereas an Activity Log Alert might have the operation that occurred

With the common alert schema, you can have standardized routing logic across alert types by leveraging the essential fields, leaving the context fields as is for the concerned teams to investigate further.



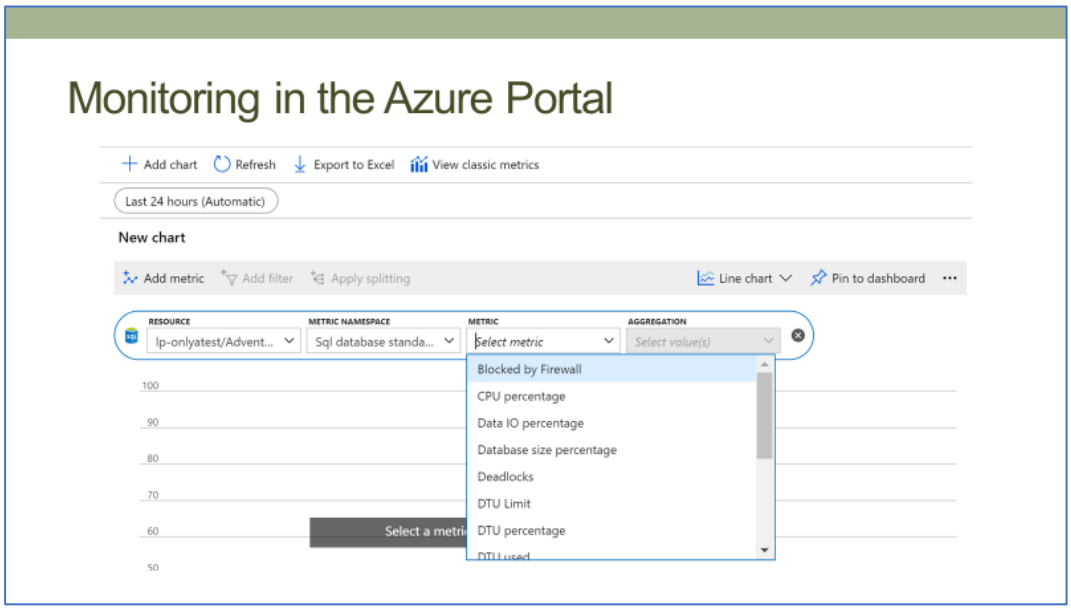
There is a rich set of monitoring charts within the Azure Portal for monitoring the performance of our Azure SQL Databases. These include:

Overview – Resource Utilization

This chart shows the consumption of resources over a configurable period. The chart can show the Max, Min or Avg for the counter. It also shows the amount of storage currently consumed by the Azure SQL Database.

Intelligent Performance – Performance Overview

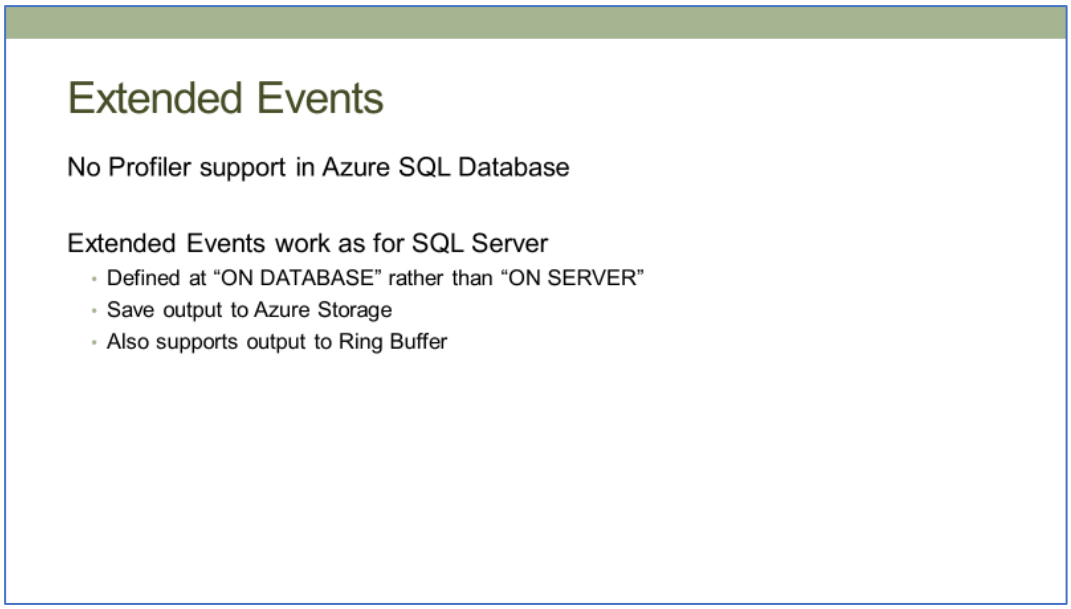
This chart shows recent Query Performance and is based on the Query Store functionality of SQL Server. You can drill further in to these charts to get additional insights.



Monitoring – Metrics

This chart shows a selection of performance metrics and can be used across multiple Azure services at the same time.

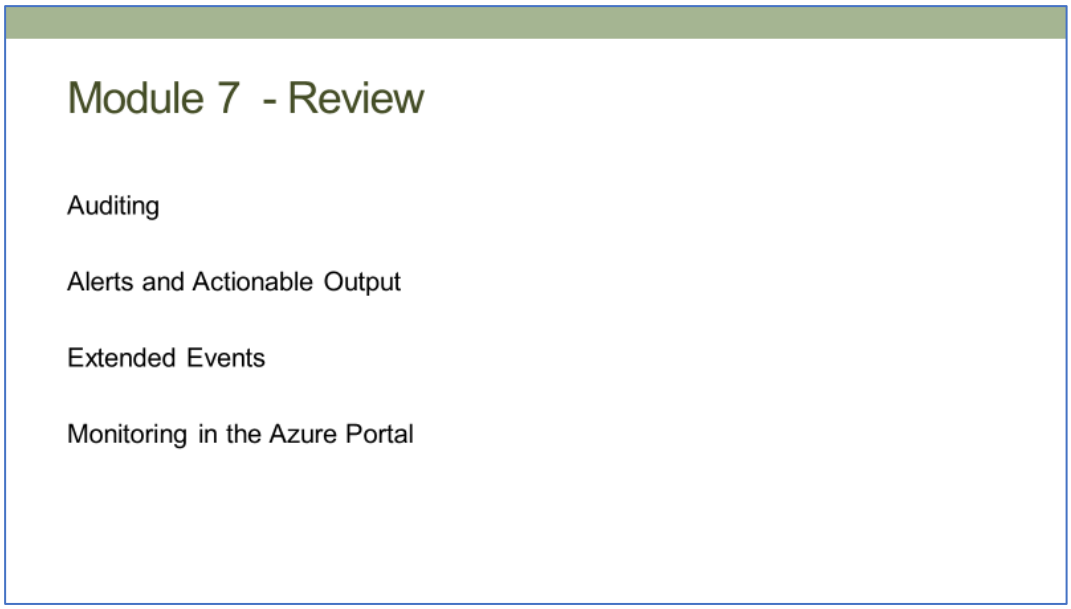
This also forms the basis for the new Alerting mechanism that was described previously.



Extended Events for Azure SQL Database is a robust subset of the Extended Events that are available for the SQL Server on premises platform.

It only varies in a couple of aspects – the scope is defined as “ON DATABASE” rather than “ON SERVER” and the only option for saving the output is to Azure Storage, although it can also be consumed from the Ring Buffer output if you only need to view results immediately (e.g. for troubleshooting).

A full discussion of Extended Events is beyond the scope of this training.



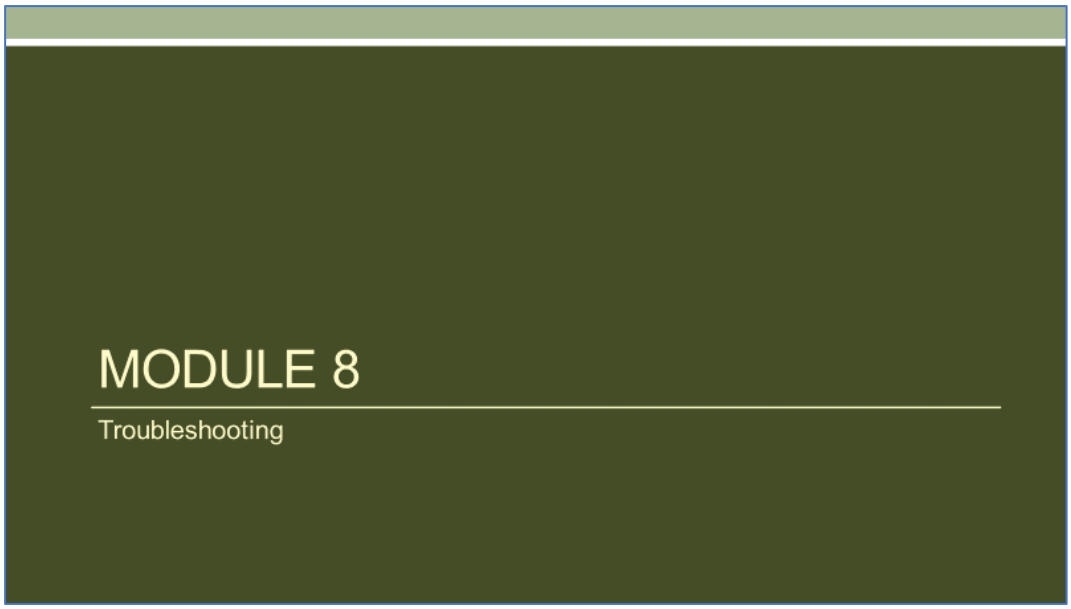
Summary

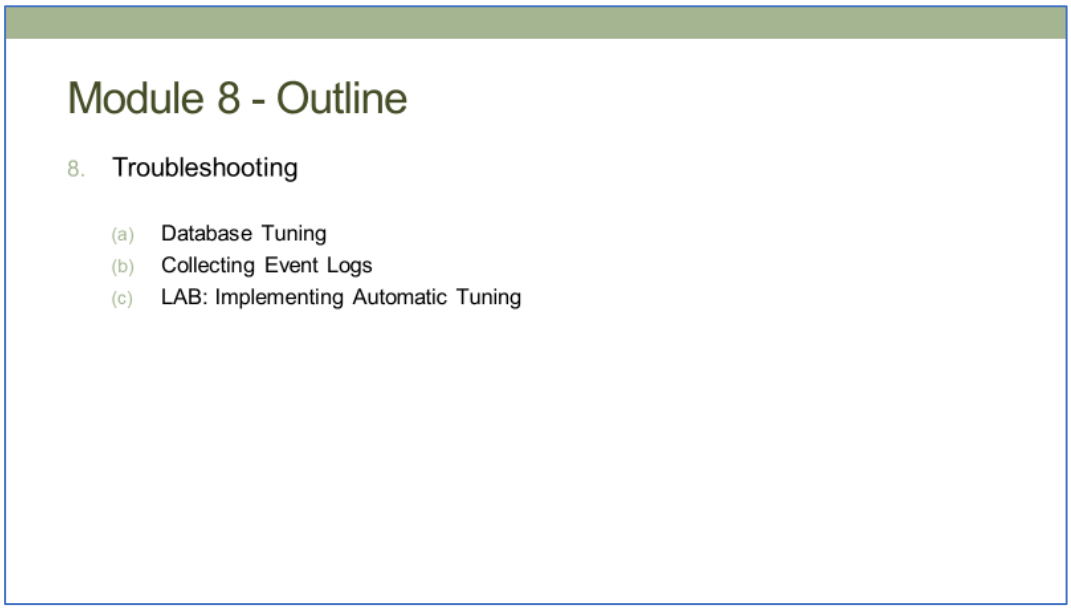
In this module we saw the methods available to review the activity on our Azure SQL Databases.

We could achieve a view of what has happened using either Audits or Extended Events.

We also saw that we could trigger Alerts based on events or metrics and that we could make these responsive by either starting an Automation Runbook or triggering a Logic App to do more complex activities.

There are also a number of inbuilt monitoring charts that are available in the Azure Portal to review performance.





Introduction

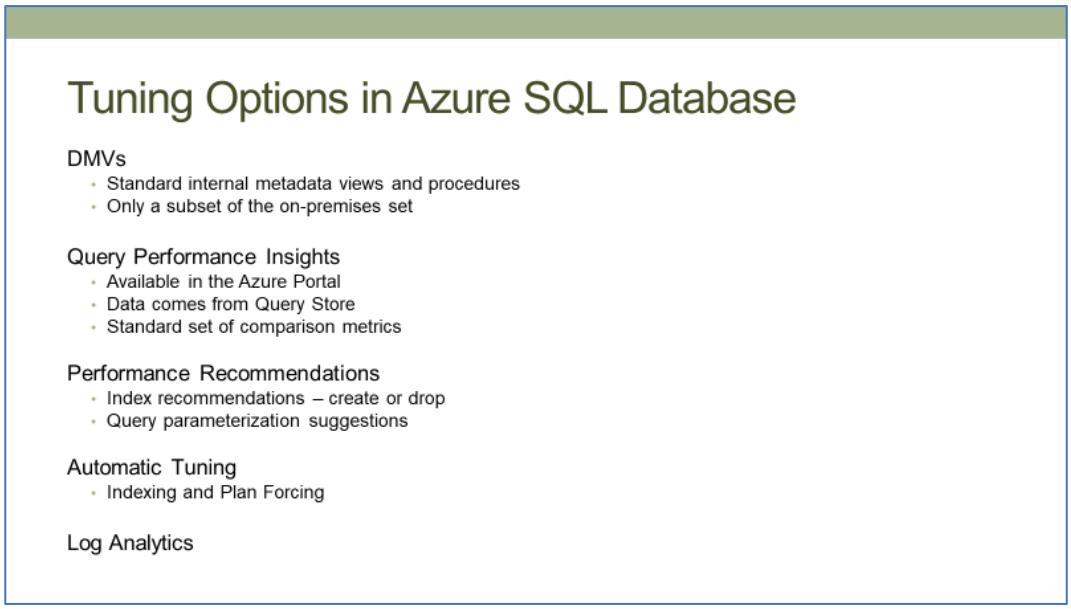
This module will discuss some processes that you can use to tune your database queries to hopefully reduce the load and your performance tier to save costs.

We will not use this as a training session for database tuning in depth but as a process to identify what is available and what you can do to start finding out the information that you need.

The tools available with Azure SQL Database include: DMVs, Query Store, Performance Recommendations, Automatic Tuning.

Resources

Log Analytics for Azure SQL Database:	https://docs.microsoft.com/en-us/azure/log-analytics/log-analytics-azure-sql
Automatic Tuning:	https://docs.microsoft.com/en-gb/azure/sql-database/sql-database-automatic-tuning
Performance Recommendations:	https://docs.microsoft.com/en-us/azure/sql-database/sql-database-advisor
Determine Plan Regression Demo:	https://blogs.msdn.microsoft.com/sqlserverstorageengine/2017/07/20/demo-identify-and-fix-plan-change-regression-in-sql-server-2017-rc1/
Automatic Index Tuning:	http://www.sqlservercentral.com/blogs/sql-and-sql-only/2018/03/30/azure-sql-database-automatic-index-tuning/
Azure SQL Analytics (Preview):	https://docs.microsoft.com/en-us/azure/azure-monitor/insights/azure-sql



Database Tuning is a complex topic and we will not be discussing the actual tuning steps in this course. What we will be focusing on is the tools and options available to help us in the tuning process.

The greatest benefit of tuning with Azure SQL Database is that we may be able to reduce the performance tier required for our database and as a result make a cost saving.

DMVs

Distributed Management Views and Functions (DMVs) have been around in SQL Server since version 2005. They are an important source of information about what is going on under the covers of SQL Server and are an invaluable asset for troubleshooting and performance tuning.

Azure SQL Database has many of these DMVs in common with the on premises version of SQL Server. These are your first port of call when working out why performance may be suffering.

One of the core DBA tasks that is not implemented automatically in Azure SQL Database is index maintenance. The relevant DMV that we can use to identify any index fragmentation issues is:

```
SELECT *
FROM sys.dm_db_index_physical_stats(DB_ID('AdventureWorksLT2016'),NULL,
NULL,NULL,'limited')
```

We can use the information from this DMV to determine whether any of our indexes should be rebuilt or reorganised to improve performance.

Query Performance Insights

Query Store was introduced with SQL Server 2016. It has become an invaluable troubleshooting assistant, particularly when identifying and resolving Query Plan regression.

The Query Performance Insights blade within Azure SQL Database provides an interface to some of the data exposed by Query Store. The charts that we see here can allow us to determine our worst performing queries by either CPU consumption, Data IO consumption or transaction log consumption. We can then start to target the queries by investigating the performance metrics. As we dive in we can see whether performance has varied over time and gain some deeper insights.

Performance Recommendations

The performance recommendations track the usage in your database and uses AI analytics to provide advice on:

Area	Scope
Optimize the layout of non-clustered indexes	Create missing indexes to improve query performance, and drop duplicate indexes to save on disk space.
Fix database schema issues	Covers situations where database schema and query definitions do not match, causing queries to fail.
Fix database parameterization issues	Covers situations where lack of proper query parameterization leads to excessive resource usage for query compilation.

Automatic Tuning

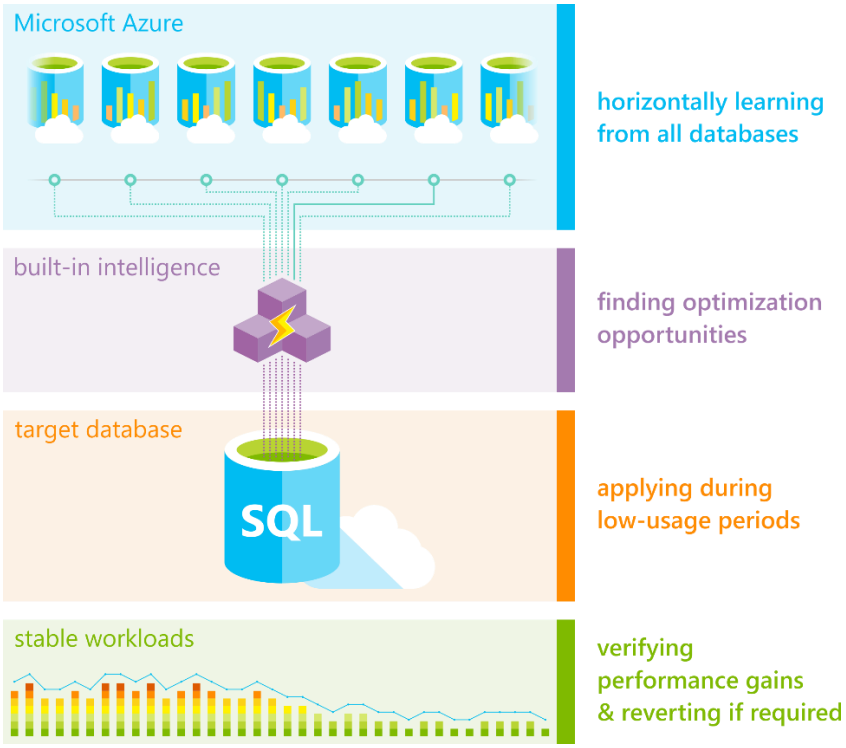
When this functionality is enabled, Azure SQL Database will monitor the database queries and perform automatic fixes to resolve them. However, it continues to monitor to ensure that the fix was valid and if not, it will back it out.

Currently this can provide the following:

Plan Forcing – where query plan regression is detected, then the last good plan can be forced to ensure consistent performance.

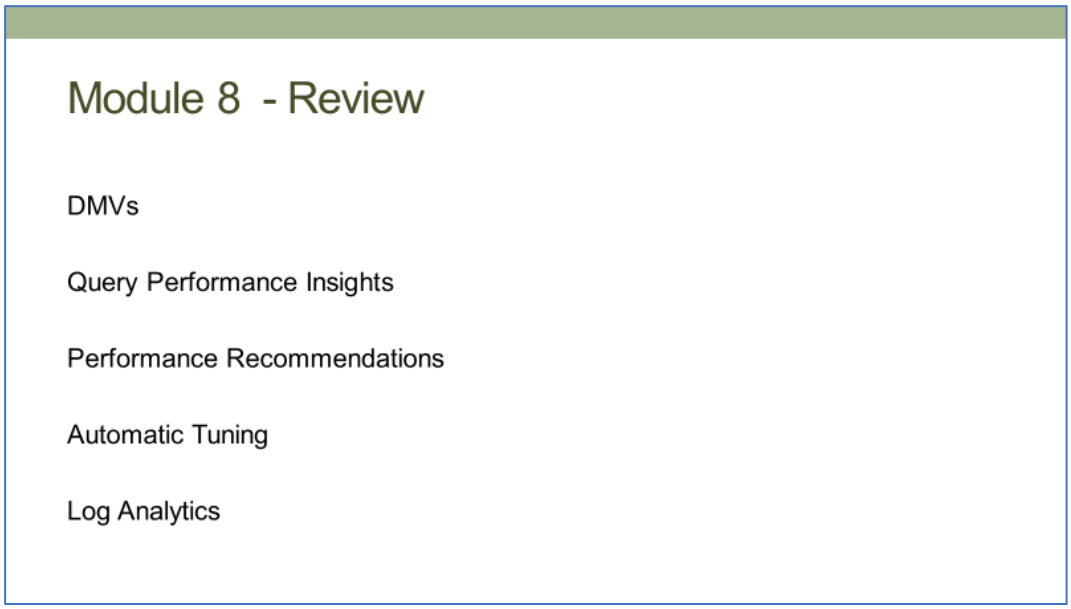
Create Index – if any missing indexes are identified in the queries, the index can be automatically created. It will be monitored for effectiveness and dropped if it didn't result in improvements.

Drop Index – duplicate and redundant indexes can be identified and removed from the database.



Log Analytics

This provides a destination for some of the Event and Metrics that we saw previously. Log Analytics has its own query syntax and is a powerful search engine across the data that it receives. This is an ideal tool for determining patterns in your data over a longer period to identify previously unknown trends.



Summary

In this module we saw some of the tools and processes that we could use to assist with our tuning efforts.

The inbuilt system views that expose the internals of SQL Server are still available in Azure SQL Database, although they are cut down in some cases to remove OS or disk-based information. The scope is also slightly different – we tend to look only at Database Scoped information.

The Azure Portal also provides some graphical tools to allow us to drill in and find our worst performing queries. We can use this manually to help but the Automatic Tuning can also assist. This functionality can detect Query Plan regressions and force a good plan choice. It can also identify, implement, monitor and confirm or remove additional indexes that it detects may be of benefit. All of these actions are reported to you in the Performance Overview blade.

The database activity can also be pushed across into Log Analytics. This provides a rich set of querying options and links through into OMS which is Microsoft's hybrid management suite.