**Managing Data in Azure SQL Database**

- Introduction / Overview of SQL Database.
- Azure SQL Managed Instance
- Comparing SQL Azure Database to Azure / On-Premise SQL Server.
- Creating and Using SQL Server and SQL Database Services.
- Azure SQL Database Tools.
- Planning the Deployment of an Azure SQL Database
- Elastic Pools.
- Export and Import of Database using .bacpac
- Azure AD Authentication for Azure SQL Database server
- Dynamic Data Masking

## Azure SQL Database Introduction

- Azure gives two options to run SQL Server workloads

  Paas :Azure SQL Database(DbaaS)

  Iaas:SQL Server on Azure VMs i.e SQL Server inside fully managed VM.
- **Azure SQL Database** is a cloud-based relational database **service** that is built on **SQL Server technologies**. It supports T-SQL commands, tables, indexes, views, primary keys, stored procedures, triggers, roles, functions etc.
- SQL Database delivers predictable performance, scalability with no downtime, business continuity and data protection—all with **near-zero administration**. You can focus on rapid app development and accelerating your time to market, rather than managing virtual machines and infrastructure.
- Because it's based on the SQL Server engine, SQL Database **supports existing SQL Server tools, libraries and APIs,** which makes it easier for you to move and extend to the cloud.
- It is available in two purchasing models DTU and vCore.
- SQL databases is available in **Basic, Standard/General Purpose, Premium/Business Critical and Hyperscale** *service tiers*. Each service tier offers different levels of performance and capabilities to support lightweight to heavyweight database workloads. You can build your first app on a small database for a few bucks a month, then change the service tier manually or programmatically at any time as your app goes viral worldwide, **without downtime to your app** or your customers.

**Benefits of SQL Database**

- **High Availability** - For each SQL database created on Windows Azure, there are **three** replicas of that database.

- **On Demand** – One can quickly provision the database when needed with a few mouse clicks.
- **Reduced management overhead** - It allows you to extend your business applications into the cloud by building on core SQL Server functionality while letting Microsoft Azure support staff handle the maintenance and patching tasks.

**SQL Database top features:**

- Tables, views, indexes, roles, stored procedures, triggers, and user defined functions
- Constraints
- Transactions
- Temp tables
- Basic functions (aggregates, math, string, date/time)
- Constants
- Cursors
- Index management and index rebuilding
- Local temporary tables
- Reserved keywords
- Statistics management
- Table variables
- Transact-SQL language elements such as create/drop databases, create/alter/drop tables, create/alter/drop users and logons

**The following features of SQL Server are NOT SUPPORTED in SQL Database**

- Windows Authentication (Azure AD Authentication is now Supported)
- Not all T-SQL Commands Supported
- Access to System Tables
- Common Language Runtime (CLR)
- Database file placement
- Database mirroring
- Distributed queries
- Distributed transactions
- Filegroup management
- Global temporary tables
- Support for SSIS (instead use Data Factory), SSAS (Separate Service), SSRS
- Support for Replication or SQL Server Service Broker

**SQL Database Deployment Options**

1. Single Database:It is Isolated single Database .It has its own guaranteed compute,memory and storage.
2. Elastic Pool:Collection of single databases with fixed set of of resources such as CPU

and Memory,shared by all databases in pool.

3. Managed Instance:Set of databases which can be used together.

**Azure SQL Database Purchasing Model**

There are two purchasing models DTU and vCore

1. DTU:
    - DTU stands for *Database Transaction Unit*, and is a combined measure of compute, storage, and IO resources.
    - DTU based model is not supported for managed instance.
2. vCores:
    - vCores are *Virtual cores*, Provides Higher compute, memory, IO, and storage limits and  give you greater control over the compute and storage resources that you create and pay for..
    - vCore model enables you to configure resources independently, gives option to choose between generation of hardware,no of cores,memory and storage size.

**Azure SQL Database Service Tiers**

- DTU and vCore based purchasing models available in three Service Tiers.
- DTU based model is available in Three service tiers:Basic,Standarad,Premium
- vCore based model is available in Three Service Tiers:General Purpose,Business Critical,Hyperscale.

## Azure SQL Managed Instance

1. It is a new deployment model of Azure SQL Database, providing near **100% compatibility** with the latest SQL Server on-premises (Enterprise Edition) Database Engine.
2. It provides a **native virtual network (VNet)** implementation that addresses common security concerns, and a business model favorable for on-premises SQL Server customers.
3. Classic on-prem application with complex environment and require SQL CLR,SQL Server Agent,Cross database queries can migrate to cloud with this model.
4. Managed Instance allows existing SQL Server customers to **lift and shift** their on-premises applications to the cloud with minimal application and database changes.
5. Managed Instance preserves all **PaaS capabilities** (automatic patching and version updates, automated backups, high-availability), that drastically reduces management overhead and administrator activities.

Visit: Azure Portal → Create a resource → Azure SQL → Create

**Comparison**

**Comparison**

| **Azure SQL Database (Logical server)** | **SQL Managed Instance** | **SQL Server on VM** |
|---|---|---|

| | | |
|---|---|---|
| PAAS Service | PAAS Service | IAAS Service |
| The most commonly used SQL Server features are available. | Near-100% compatibility with SQL Server. on-premises. | Fully compatible with on-premises physical and virtualized installations. |
| You can provision a *single database* in a dedicated, managed (logical) server; or you can use an *elastic pool* to share resources across multiple databases and take advantage of on-demand scalability. | Each managed instance can support multiple databases.<br>Additionally, *instance pools* can be used to share resources efficiently across smaller instances. | SQL Server instances are installed in a virtual machine.<br>Each instance can support multiple databases. |
| 99.995% availability guaranteed. | 99.99% availability guaranteed. | Up to 99.99% availability. |
| Latest stable Database Engine version. | Latest stable Database Engine version. | Fixed, well-known database engine version.<br>All SQL Server Features are avaiable |
| Fully automated updates, backups, and recovery. | Fully automated updates, backups, and recovery. | You must manage all aspects of the server, including operating system and SQL Server updates, configuration, backups, and other maintenance tasks. |
| Migration from SQL Server might be hard. | Easy migration from SQL Server. | Easy migration from SQL Server on-premises. |
| Built-in | Built-in | You need to implement your own High-Availability solution. |
| Ability to assign necessary resources (CPU/storage) to individual databases.<br>Online change of resources (CPU/storage). | Online change of resources (CPU/storage). | There is a downtime while changing the resources (CPU/storage) because VM needs to resized and that restarts VM |

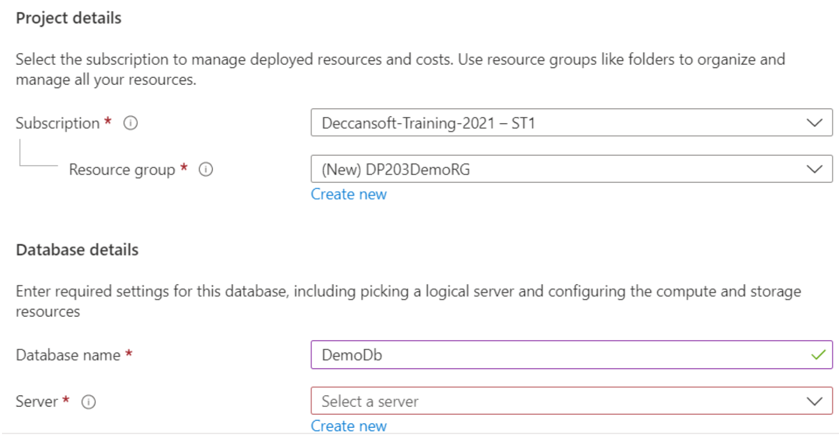
## Creating SQL Database

With Windows Azure SQL Database you can quickly create database solutions that are built on the SQL Server database engine. We can create a new SQL database in Windows Azure and then configure it later. We can decide whether to use an existing SQL database server or create a new one when you create your new database. We can also import a saved database from Binary Large Object (BLOB) storage into SQL Database.

***Azure SQL logical server***

- You need to create *Azure SQL logical server* When you create your first Azure SQL database.
- *Consider Azure SQL logical server as, nothing but* administrative container for your databases(SQL Database, Warehouse Database).
- You can control logins, firewall rules, and security policies through the logical server.
- You can also override these policies on each database within the logical server.

### Lab 1 :Creating an Azure SQL Database Instance by using the Azure Portal



**Project details**

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

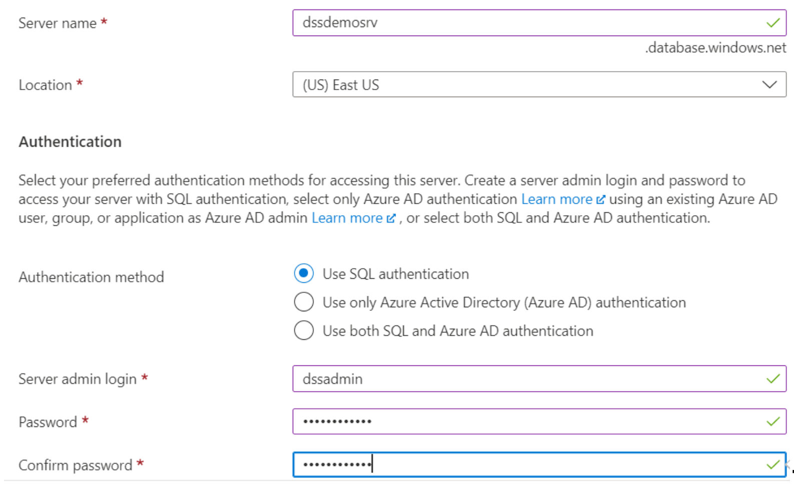
| | |
|---|---|
| Subscription * | Deccansoft-Training-2021 – ST1 |
| Resource group * | (New) DP203DemoRG |
| | Create new |

**Database details**

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

| | |
|---|---|
| Database name * | DemoDb |
| Server * | Select a server |
| | Create new |

→Create new


Server name * dssdemosrv .database.windows.net

Location * (US) East US

**Authentication**

Select your preferred authentication methods for accessing this server. Create a server admin login and password to access your server with SQL authentication, select only Azure AD authentication Learn more using an existing Azure AD user, group, or application as Azure AD admin Learn more , or select both SQL and Azure AD authentication.

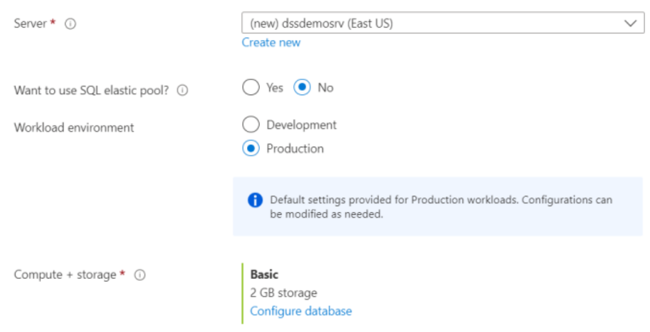
Authentication method: [✓ Use SQL authentication] / Use only Azure Active Directory (Azure AD) authentication / Use both SQL and Azure AD authentication

Server admin login * dssadmin

Password * ••••••••••••

Confirm password * ••••••••••••

→OK


Server * (new) dssdemosrv (East US)
Create new

Want to use SQL elastic pool? Yes [✓ No]

Workload environment: Development [✓ Production]

Default settings provided for Production workloads. Configurations can be modified as needed.

Compute + storage * Basic
2 GB storage
Configure database

→Review+Create→Create

## Azure SQL Database Tools

One of the advantages of SQL databases in Azure is the ability to use many monitoring tools that you use for on-premises databases.

A TDS endpoint is exposed for each logical server in SQL Database. This allows you to use SQL Server Management Studio with SQL Database in the same way you will use it with SQL Server standalone.

**Using SQL Server Management Studio:**

1. Start SQL Server Management Studio locally
2. In Connect dialog, provide
    a. Server name= "**dssdemoserver.database.windows.net"**
    b. **Change Authentication = SQL Server Authentication**
    c. Login = "DSSAdmin"
    d. Password = "Password@123"
    e. Connect
3. **This will give error**. From the error dialog note the IP address eg: 49.12.12.4

4. Configure firewall settings on SQL Server using the Azure Portal
    a. Azure Portal → Select Sql Server → Settings → **Firewall**
    b. Add Local IP OR

        Click Add Client IP**.**
        i. Rule Name = "Allowed IP".
        ii. Start IP = 49.12.12.0
        iii. End IP = 49.12.12.5
    c. Click Save
5. Return back to SSMS and try to connect again. (It might take upto 5 mins after allowing the IP in firewall)

To Restrict a given ip or a range of IP address for a particular database

```
-- Create database-level firewall setting for only IP 0.0.0.4
EXECUTE sp_set_database_firewall_rule N'Example DB Setting 1', '49.12.12.4', '49.12.12.4';

-- Update database-level firewall setting to create a range of allowed IP addresses
EXECUTE sp_set_database_firewall_rule N'Example DB Setting 1', '49.12.12.0', '49.12.12.100';
```

Note: If you specify an IP address range in the database-level IP firewall rule that's outside the range in the server-level IP firewall rule, only those clients that have IP addresses in the database-level range can access the database.

## Planning the deployment of an Azure SQL Database

- Azure SQL Database has three deployment options:Single Database,elastic Pool,Managed Instance
- Your first step should be to determine whether any service **specific functional limitations** would require that you choose an IaaS-based implementation of SQL Server or another relational database management system.
- When you plan an Azure SQL Database deployment, your primary consideration should be the database's **intended workload**.
- Also, you should consider the scalability limits of Azure SQL Database. These might include maximum **supported database size or performance**, with respect to transactional throughput, maximum concurrent requests, maximum sessions, and maximum concurrent logins.

### Azure SQL Database Purchasing Model

There are two purchasing models DTU and vCore

1. DTU:
    - The relative measure of a database's ability to handle resource demands is expressed in **Database Transaction Units** (DTUs). **DTUs provide a way to describe the relative capacity of a performance**

**level based on a blended measure of CPU, memory, reads, and writes**.SO DTU is a combined measure of compute, storage, and IO resources

- Think of the DTU model as a simple, preconfigured purchase option as it provides fixed combination of CPU,Memory and IO.
- DTU based model is not supported for managed instance.

2. vCores:
   - vCores are *Virtual cores*, Provides Higher compute, memory, IO, and storage limits and  give you greater control over the compute and storage resources that you create and pay for..
   - vCore model enables you to configure resources independently, gives option to choose between generation of hardware,no of cores,memory and storage size.
   - Example: With the vCore model you can increase storage capacity but keep the existing amount of compute and IO throughput.
   - This purchasing model provides a choice between a provisioned compute tier and a serverless compute tier.
   - With the provisioned compute tier, you choose the exact amount of compute resources that are always provisioned for your workload. With the serverless compute tier, you specify the autoscaling of the compute resources over a configurable compute range.
   - With this compute tier, you can also automatically pause and resume the database based on workload activity.
   - The vCore unit price per unit of time is lower in the provisioned compute tier than it is in the serverless compute tier.
   - vCore is supported for all deployment options,except **hyperscale service tier** which is supported only for single database.

**Azure SQL Database Service Tiers**

DTU and vCore based purchasing models  available in three Service Tiers.

DTU based service tier:

| **Service tier features** | **Basic** | **Standard** | **Premium** |
|---|---|---|---|
| DTU | 5 | 10-3000 | 125-4000 |
| **Database backup retention period (Point-In-time restore)** | 7 days | 35 days | 35 days |
| **IO Latency** | 5-10 ms | 5-10 ms | 2 ms |

| | | | |
|---|---|---|---|
| **Maximum individual database size** | 2GB | 1TB | 4TB |
| **Max in-memory OLTP storage** | N/A | N/A | 1-32 GB |

Note:All service tier offers 99.99% uptime SLA

**Vcore based service tier**

| **Service tier features** | General Purpose | Business Critical | Hyperscale |
|---|---|---|---|
| **Maximum individual database size** | **provisioned compute**:<br>5 GB – 4 TB<br>**Serverless compute**:<br>5 GB - 3 TB | **provisioned compute**:<br>5 GB – 4 TB | Supports up to 100 TB of storage. |
| **vCore** | 10-3000 | 125-4000 | |
| **Availability** | 99.99%<br>1 replica, no read-scale replicas | 99.995%<br>3 replicas, 1 **read-scale replica,**<br>zone-redundant high availability (HA) | 1 read-write replica, **plus 0-4 read-scale replicas** |
| **Database backup retention period (Point-In-time restore)** | 7-35 days (7 days by default) | 7-35 days (7 days by default) | Snapshot-based backups in Azure remote storage. Restores are fast and aren't a size-of-data operation (taking minutes rather than hours or days). |
| **Max in-memory OLTP storage** | N/A | Supported | N/A |

**Choosing service Tier**

The following table provides examples of the tiers best suited for different application workloads.

| Service tier | Target workloads |
|---|---|
| **Basic** | Best suited for a small database, supporting typically **one single active operation** at a given time. Examples include databases used for development or testing, or small-scale infrequently used applications. For infrequent access and less demanding workloads. |
| **Standard/General Purpose** | Suited for most of the **generic ,budget oriented** workloads.The go-to option for cloud applications with low to medium IO performance requirements, supporting **multiple concurrent queries**. Examples include web applications. |

| | |
|---|---|
| **Premium/Business Critical** | Designed for high transactional volume with high IO performance requirements, supporting many concurrent users. Suited for applications that require **low-latency** responses from the underlying SSD storage (1-2 ms in average), **fast recovery** if the underlying infrastructure fails, or need reporting and read only analytic queries that can be redirected to the free-of-charge secondary read-only replica..Examples are databases supporting mission critical applications. |
| **Hyperscale** | Suited for large databases as it supports database size upto 100TB. Also suited for smaller databases, but require fast **vertical and horizontal compute scaling, high performance, instant backup, and fast database restore.** |

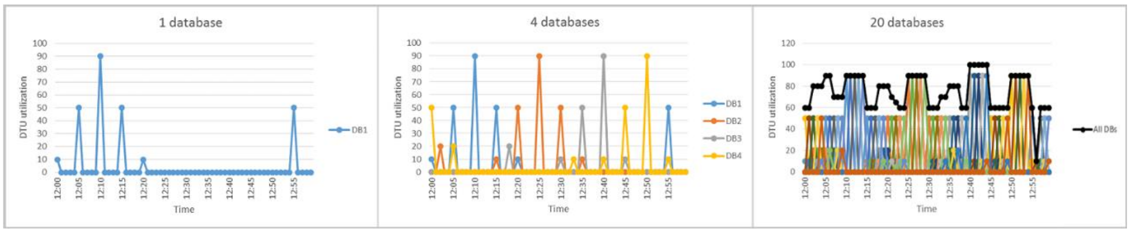
In general, you have two choices, including:

1. A traditional approach, which provides a dedicated set of resources for each database. It does this by assigning a pricing tier to it, which determines its sizing and performance characteristics.
2. A second approach, introduced in Azure SQL Database V12, which allows you to distribute resources among multiple databases that are hosted on the same logical server by combining them into **elastic database pools.**

**Elastic Pools:**

- Elastic pools provide a **simple cost effective solution** to manage the performance goals for **multiple databases** (hosted on the same logical server) that have widely **varying and unpredictable** usage patterns.
- **elastic DTUs** (eDTUs) are used elastic databases in an elastic pool.
- A pool is given a set number of eDTUs, for a set price. Within the pool, individual databases are given the flexibility to auto-scale within set parameters.
- Provisioning resources for the entire pool rather than for single databases simplifies your management tasks.
- Under heavy load, a database can consume more eDTUs to meet demand. Databases under light loads consume less, and databases under no load consume no eDTUs.
- Additional eDTUs can be added to an existing pool with no **database downtime** or no impact on the databases in the elastic pool. Similarly, if extra eDTUs are no longer needed they can be removed from an existing pool at any point in time.
- You can add or subtract databases to the pool. If a database is predictably under-utilizing resources, move it out.

**Which databases go in a pool?**



- Databases that are great candidates for elastic pools typically have periods of activity and other periods of inactivity. In the example above you see the activity of a single database, 4 databases, and finally an elastic pool with 20 databases.
- Databases with varying activity over time are great candidates for elastic pools because they are not all active at the same time and can share eDTUs.
- Not all databases fit this pattern. Databases that have a more constant resource demand are better suited to the Basic, Standard, and Premium service tiers where resources are individually assigned.
- 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.**

**Cost of Single Database** = Database count * Cost of Each DTU * Number of DTU

**Cost of Elastic Pool** = Cost of eDTU * Number of eDTU **=** 1.5 * Cost of each DTU * Number of eDTU

**Sizing an elastic pool:**

The best size for a pool depends on the aggregate eDTUs and storage resources needed for all databases in the pool. This involves determining the larger of the following:

- Maximum DTUs utilized by all databases in the pool.
- Maximum storage bytes utilized by all databases in the pool.

SQL Database automatically evaluates the historical resource usage of databases in an existing SQL Database server and recommends the appropriate pool configuration in the Azure portal.

**Creating a Pool and adding database to it.**

1. Azure Portal → **SQL Servers** → Server blade → New Pool
2. Name = DemoPool
3. Pricing tier = Standard Pool (The pool's pricing tier determines the features available to the elastic databases in the pool, and the maximum number of eDTUs (eDTU MAX), and storage (GBs) available to each database.)
4. Configure the Pool → Specify Elastic database pool settings and in blade on top click **Add to Pool** to add database to the pool
5. Also Per database settings can be specified for eDTU max and min.

**Creating a Pool and adding database to it.**

1. Azure Portal→Create Resource→Elastic Pool→Select SQL Elastic pool→+New→
2. Elastic Pool Name:DemoPool
3. Pricing tier = Standard Pool (The pool's pricing tier determines the features available to the elastic databases in the pool, and the maximum number of eDTUs (eDTU MAX), and storage (GBs) available to each database.)
4. Configure the Pool → Specify Elastic database pool settings and in blade on top click **Add to Pool** to add database to the pool.
5. Also Per database settings can be specified for eDTU max and min.

*Note:If you select Vcore model under pool settings you can set Vcore and Data max size .Under Per datbase settings you can specify vCores.*

You can add or remove database to pool by going to **configuration** section.

*Refer:*

https://docs.microsoft.com/en-us/azure/azure-sql/database/elastic-pool-overview

## Export and Import of Database using .bacpac

In Azure SQL Database, you **cannot** directly use the database and transaction log backup capabilities of SQL Server. Historically, this was remediated by periodically **exporting a copy** of each database that you want to protect, and storing the copy in a .**bacpac** file in a storage account. In the event of a SQL database or server failure, you could then create a new SQL database server, if necessary, and **import the copy** of the database from the exported file.

**Export of Database:**

- When you need to export a database for archiving or for moving to another platform, you can export the database schema and data to a BACPAC file.
- A BACPAC file is a ZIP file with an extension of BACPAC containing the metadata and data from a SQL Server database.
- A BACPAC file can be stored in Azure blob storage or in local storage in an on-premises location and later imported back into Azure SQL Database or into a SQL Server on-premises installation.
- If you are exporting to blob storage, **the maximum size of a BACPAC file is 200 GB**. To archive a larger BACPAC file, export to local storage.

- For an export to be transactionally consistent, you must ensure either **that no write activity** is occurring during the export, or that you are exporting from a transactionally consistent **copy** of your Azure SQL database.

**Steps to Export:**

1. Azure Portal → SQL databases → Select the Database
2. **Copy Database:** Azure Portal → SQL databases → Select the Database → Click Copy in database blade → Provide the required details → OK.
    a. Can be either of same or different server
    b. Service Tier can be changed.
3. Goto to **Copy of database** → Click **Export** in database blade → Provide the required details including Storage Account, Server Admin Login/Password → OK

Note: The length of time the export will take depends on the size and complexity of your database, and your service level. You will receive a notification on completion.

4. **Monitor the progress of the export operation**
   **Azure Portal → Click SQL servers → click the server containing the original (source) database you just archived → Scroll down to Operations → click** Import/Export history**:**

**Note: The newest versions (v17 / 2017) of SQL Server Management Studio also provide a wizard to export an Azure SQL Database to a bacpac file.(Database→RC→Tasks→Export Data Tier Application)**

**Import a BACPAC file to create an Azure SQL database**

1. Azure Portal → **SQL Servers** → In SQL Server blade → **Import database**
2. Click **Storage** and select your storage account, blob container, and .bacpac file and click **OK**
3. Select the pricing tier for the new database and click **Select**
4. Enter a **Database Name** for the database you are creating from the BACPAC file.
5. Choose the authentication type and then provide the authentication information for the server.
6. Click **Create** to create the database from the BACPAC.

## Azure AD Authentication for Azure SQL Database server

- Azure AD Authentication is a mechanism of connecting to Azure SQL Database, managed instance by using identities in Azure Active Directory (Azure AD).

- Azure AD authentication uses contained database users to authenticate identities at the database level
- Azure AD supports token-based authentication for applications connecting to SQL Database.

**Traditional Login and User Model**

- Traditionally for users to have access to database, the master database must have a login that matches the connecting credentials and the login must be able to be mapped to a database user in the user database.
- connection to the user database has a dependency upon the login in the master database, and this limits the ability of the database to be moved to a different hosting SQL Server or Azure SQL Database server.

**Contained Database User Model**

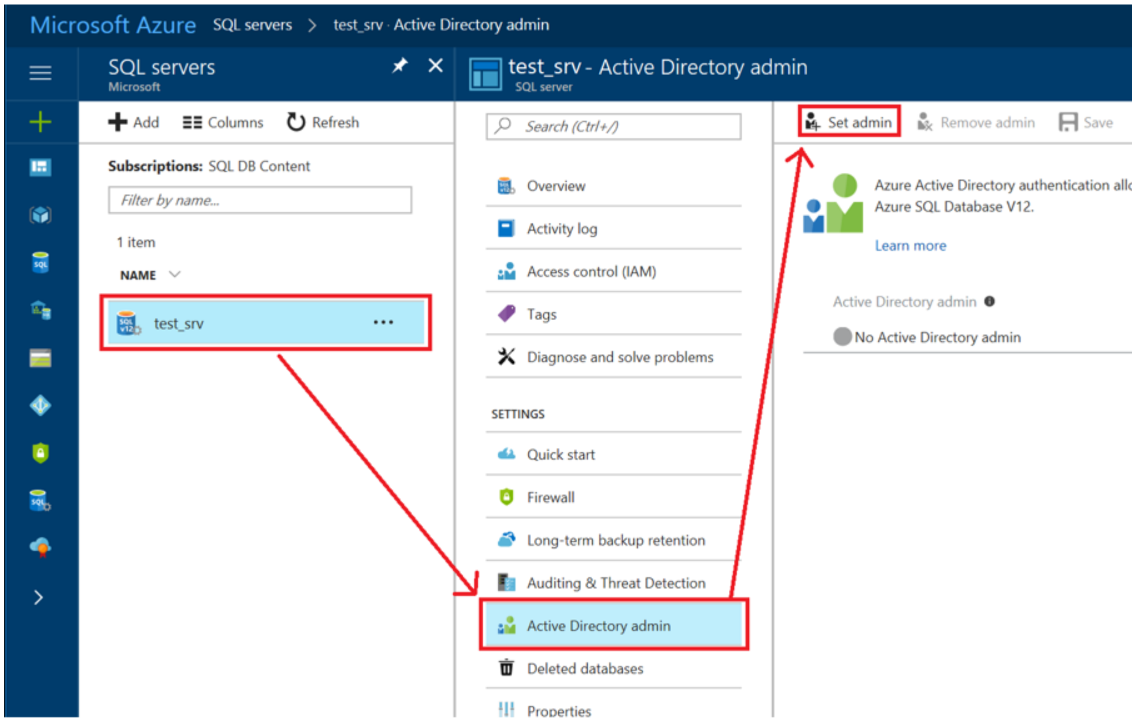
- In the contained database user model ,the authentication process occurs at the user database, and the database user in the user database does not have an associated login in the master database.
- To connect as a contained database user, the connection string must always contain a parameter for the user database so that the Database Engine knows which database is responsible for managing the authentication process.
- Azure sql database support Azure Active Directory identities as contained database users.
- When using Azure Active Directory authentication, connections from SSMS can be made using Active Directory Universal Authentication.
- The ADO.NET provider for SQL Server, SqlClient, now supports setting the AccessToken property to authenticate SQL Server connections using Azure Active Directory. In order to use the feature, you can obtain the access token value using Active Directory Authentication Library for .NET, contained in the Microsoft.IdentityModel.Clients.ActiveDirectory NuGet package.

**Configure and manage Azure Active Directory authentication with SQL**

1. Create an Azure AD and populate it with users and groups
2. Associate your Azure subscription to Azure Active Directory by making the directory a trusted directory for the Azure subscription hosting the database.

3. **Create an Azure AD administrator for Azure SQL server**

   I. Azure Portal → SQL Servers → Select the <Server>
   II. On **SQL Server** page, select **Active Directory admin**.
   III. In the **Active Directory admin** page, select **Set admin.**



IV. In the **Add admin** page, search for a user, select the user or group to be an administrator, and then select **Select**.

*Note*

- *The Active Directory admin page shows all members and groups of your Active Directory. Users or groups that are grayed out cannot be selected because they are not supported as Azure AD administrators. (See the list of supported admins in the **Azure AD Features and Limitations** section of Use Azure Active Directory Authentication for authentication with SQL Database or SQL Data Warehouse.)*
- *Removing the Azure Active Directory administrator for Azure SQL server **prevents any Azure AD authentication user** from connecting to the server. If necessary, unusable Azure AD users can be dropped manually by a SQL Database administrator.*

## 1. Create contained database users in your database mapped to Azure AD identities

Execute the following commands to create a contained **database user**

```
CREATE USER [abc@Xyz.com] FROM EXTERNAL PROVIDER;
```

To create a contained database user representing an **Azure AD domain group**, provide the display name of a security group:

```
CREATE USER [ADGroup1] FROM EXTERNAL PROVIDER;
```

## 2. Grant required permissions to user using Grant command

```
GRANT SELECT,INSERT ON EMPLOYEE to abc@Xyz.com
```

## Dynamic Data Masking

- Dynamic data masking (DDM) limits sensitive data exposure by masking it to non-privileged users. It can be used to greatly simplify the design and coding of security in your application.
- Dynamic data masking is a great feature for both on-premise SQL Server (from SQL Server 2016) and Azure SQL Database as well. This feature can help users to secure their critical data elements without making any change at physical level.
- All the unprivileged users can only see masked data and don't have access to actual values since masking rules are applied in the query results
- Dynamic data masking is easy to use with existing applications.
- As an example, a call center support person may identify callers by several digits of their social security number or credit card number. Social security numbers or credit card numbers should not be fully exposed to the support person. A masking rule can be defined that masks all but the last four digits of any social security number or credit card number in the result set of any query

```
Create table DemoTable
        (ID Int, PersonName varchar (100),
        Age int,
        EmailAddress varchar(120),
        CreditCardNumber varchar(19),
        SocialSecurityNumber varchar(11))
INSERT INTO DemoTable Values (1, 'Sandeep Soni',43,'sandeep@abc.com','1234-5678-4321-8765','123-45-6789')
SELECT * FROM DemoTable --Result will not be masked
```

**Masking Functions**

SQL Server provides four built in functions to mask data in SQL tables. These functions are as follows:

1. default():Full masking according to the data types of field.

   For string data types, use XXXX or fewer Xs if the size of the field is less than 4 characters

   For numeric data types use a zero value

   For date and time data types use 01.01.1900

   Example: Alter Table DemoTable
   ```
           Alter Column PersonName varchar (100)  MASKED WITH (FUNCTION='default()')
   ```
2. email():**Masking method, which exposes the first letter and replaces the domain with XXX.com**

Example: aXX@XXXX.com

Example: Alter Table DemoTable
Alter Column EmailAddress varchar (120) MASKED WITH (FUNCTION='email()')

3. random(): **Masking method, which generates a random number** according to the selected boundaries. If the designated boundaries are equal, then the masking function is a constant number. Present as "**Random number"** in portal

   Example: Alter Table DemoTable
   Alter Column age int MASKED WITH (FUNCTION ='random(1,20)')

4. Partial(): Masking method that exposes the first and last letters and adds a custom padding string in the middle. If the original string is shorter than the exposed prefix and suffix, only the padding string is used.Present as "

   **"Custom text"** in portal.

   Example: prefix[padding]suffix:3[X-X-X-X]1 →123X-X-X-X9

   Example: Alter Table DemoTable
   Alter Column SocialSecurityNumber varchar(11) MASKED WITH (FUNCTION ='partial(2,"XXXXX",3)')

5. Credit card: **Masking method, which exposes the last four digits of the designated fields** and adds a constant string as a prefix in the form of a credit card.

   Example: XXXX-XXXX-XXXX-1234

Note:Masking functions available in portal:Default,Creditcard,Email,Randum number,Custom text

**Dropping Mask**

ALTER TABLE DemoTable

ALTER COLUMN SocialSecurityNumber DROP MASKED;

**To Mask Data using portal**:

Azure Portal → Select Database → **Dynamic Data Masking** → + Add Mask (Look at Recommended fields to mask)

**Note: We can exclude the SQL users from masking and administrators are always excluded**

Now execute the command again:

```
SELECT * FROM DemoTable
```

We can see that data is still visible as inserted. There is no change in data behavior and the data doesn't mask. The reason for this behavior is user permission. In the current scenario, my ID has db_owner permission and has full access to the data.

**To understand the behavior of mask functions and masked data**, we will create a new database user TestMaskUser (without login) and will grant select permission on the TestDDM table to the newly created database user.

```
CREATE USER TestMaskUser WITHOUT LOGIN;

GRANT SELECT ON DemoTable TO TestMaskUser;
```

Now, we will change the context of the query execution and review the TestDDM data table.

```
EXECUTE AS USER = 'TestMaskUser';

SELECT * FROM DemoTable;

REVERT;
```

**Grant and Revoke UNMASK Permission**

UNMASK permission,when granted to a user, the user can see the original values in a table.

```
GRANT UNMASK TO TestMaskUser;

REVOKE UNMASK TO TestMaskUser;
```

More: https://docs.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-2017