Agenda: Managing Data in Azure SQL Database

- Introduction/Overview of SQL Database.
- Comparing SQL Azure Database to Azure / On-Premise SQL Server.
- Creating and Using SQL Server and SQL Database Services.
- Azure SQL Database Tools.
- Migrating on premise database to SQL Azure.
- Planning the Deployment of an Azure SQL Database
- Elastic Pools.
- Monitoring Azure SQL Database
- Configure SQL Database Auditing
- Manage Business Continuity
- Export and Import of Database
- Backup and Recovery options in SQL Database
- Active GEO-Replication
- Long Term Backup Retention

Introduction

- **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.
- SQL databases is available in **Basic, Standard, and Premium** *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 Windows 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
- User-defined types
- 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 (Power BI)
- Support for Replication or SQL Server Service Broker
- Support Backup and Restore

Note that SQL Database Supports **Horizontal Scaling** and SQL Database requires **Clustered Indexes**

Topic	SQL Database (PaaS)	SQL Server on Azure VM (IaaS)
Features	Less features than box, optimized to	Full box product features, optimized for the best

	reduce costs	compatibility with existing applications and for hybrid applications
Performances	Max 4000 DTU in Premium Tier	Depends on VM SKU/Storage
DB Size	Max 4TB in Premium Tier (P15)	64TB on G-SERIES (Max Size of Disk supported by Azure)
Workload	Sizing by average usage	Sizing based on peaks
High-Availability	Built-in by platform. 99.99% high-availability SLA	Manual configuration by AlwaysOn AG. 99.95% HA SLA that covers just the Virtual Machines in an availability set
Fault-Handling	Necessary fault-handling & retry	Recommended fault-handling &retry
Locality	No co-location with application	Co-located by VMs and VNETs
Segregation	Internet exposed endpoint / VNet	Internal private endpoint
Versioning	No control on upgrades	Full control over DB upgrade
Total Cost of Operation	Very low, almost self-managed. Total cost of application = Highly minimized administration costs + software development costs + SQL Database service costs	High (as on-premises). Total cost of application = Highly minimized software development cost + administration costs + SQL Server and Windows Server licensing costs + Azure Storage costs
Administration	No full-time DBA required	Full staffed DBA required
Management	Easy to manage many DBs	Complex to manage many DBs/VMs
Scale-Out	Tools & Frameworks available	No easy scale-out
Configuration	No setup customization	Full access to OS and SQL Server instance level properties
Authentication	SQL standard authentication, Azure Active Directory authentication	SQL standard and integrated
Security	No Fixed IP, fixed 1433 port	Fixed IP possible, port can be changed
Backup	Backup files not accessible, 35 days Point-in-Time-Restore	Full control of backup files, unlimited PITR
Hybrid	No AlwaysOn AG support	Can join on-premises AlwaysOn AG topology
Cross-DB Access	NO: Distributed Transaction Coordinator (DTC), Linked Servers, USE statement, 4-parts names	YES: DTC, Linked Srv, USE, 4-parts names
Migrate Existing Apps	Moderate	Fast

Build New Apps	Fast	Moderate
Licensing	Not required	Regardless of VM size or SQL Server edition, pay per-minute licensing cost of SQL Server + Windows Server + Azure Storage cost for the VM disks unless Bring-Your-Own-License option is chosen

Azure SQL Database 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. Managed Instance allows existing SQL Server customers to **lift and shift** their on-premises applications to the cloud with minimal application and database changes.
4. Managed Instance preserves all **PaaS capabilities** (automatic patching and version updates, automated backups, high-availability), that drastically reduces management overhead and TCO.

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.

Creating an Azure SQL Database Instance by using the Azure Portal

1. Sign in to the Azure Preview Portal (https://portal.azure.com).
2. Create New SQL Server
   a. Browse → SQL Servers → Add
   b. Provide unique Server Name = dssdemoserver (all lowercase), Server Admin login =DSSAdmin, Password = User@123 and other details…
   c. Create
3. Create New SQL Database
   a. Browse → SQL Database → Add
   b. Select Server created in earlier step and other details. Database Name= mydemodatabase
   c. 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. Select Sql Database →In Essentials Section → Click on "Show database connection strings" → Copy ADO.NET Connection String and paste 'the same in notepad. Manually provide User Id and Password values.

   Eg: **Server=tcp:dssdemoserver.database.windows.net,1433;Database=mydemodatabase;User ID=DSSAdmin;Password=*User@123*;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;**
2. Start SQL Server Management Studio locally
3. 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
4. **This will give error**. From the error dialog note the IP address eg: 49.12.12.4
5. 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
6. Return back to SSMS and try to connect again. (It might take upto 5 mins after allowing the IP in firewall)

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

-- Update database-level firewall setting to create a range of allowed IP addresses
EXECUTE sp_set_database_firewall_rule N'Example DB Setting 1', '0.0.0.0', '0.0.0.6';
```

Using Visual Studio:

You can use the **SQL Server Object Explorer** to manage both individual databases and an entire server.

Test Connection in an Application:

1. Create a Console Based Application
2. Edit App.Config and add ConnectionStrings Section with the value copied earlier.
3. Edit Main as below

```
SqlConnection con = new SqlConnection();
con.ConnectionString =
System.Configuration.ConfigurationManager.ConnectionStrings["DemoDbCS"].ConnectionString;
SqlCommand cmd = new SqlCommand("Select * from Emp", con);
con.Open();
Console.WriteLine("Connection Successfully Opened");
con.Close();
```

4. Run the application.

Migrating on premise SQL Server database to Azure SQL Database

- Migration tools help in handling repetitive tasks of migrating (Ex:- Migrating columns, identity field etc)
- Consider what data to move, prepare for the move and use tools for the migration.
- Any data of a RDBMS can be moved to SQL Azure

Tools for migration of databases are:-

1) Generate SQL Script using SQL Server Management Studio.
2) Microsoft SQL Server Migration Assistant (SSMA)- It can move data from (Oracle, MySql, Sybase, IBM-db2, Microsoft Sql Server)

Migrating using SQL Script:

Step1: Generating the SQL Script

1. Open SQL Server Management Studio
2. Right click the source database. Choose **Tasks->Generate Scripts.**
3. Press **Next** on the first screen of the Wizard then choose the database and check the **Script all objects and all database objects** checkbox. Press **Next**.
4. You will need to edit the script by hand after it is generated, but, at the same time you can save some work by modifying some of the default script options. In the **Set Scripting Options** page, press the **Advanced** button. The following bullets specify the changes to the default options and the justification for doing so:

a. Set **Convert UDDTs to base types** to **True** - SQL Azure does not support User Defined Data Types. You should use this option to convert any user-defined types into their underlying base types.
b. Set **Script extended properties** to **False** - SQL Azure does not support extended properties. Therefore you do not need to script out these properties.
c. Set **Script USE DATABASE** to **False** - SQL Azure does not support the USE DATABASE command for changing database context.
d. Set **Types of data to script** to **Schema and data** - For the purposes of this lab, we want to script not only the database schema but also the data contained therein.

5. After setting all the properties press **Next**.
6. Leave the default **Script to New Query Window** radio button selected. Press **Next**.
7. Press **Finish**.

Step 2: Executing the SQL Script

1. Delete the initial part of the script file related to Create Database and Alter Database…
2. Connect to Azure Database from SSMS
3. Execute the Script.

Step 3: Create a New ASP.NET MVC Web Application and add the following to Index:

```
SqlConnection con = new SqlConnection();
con.ConnectionString =
System.Configuration.ConfigurationManager.ConnectionStrings["DemoDbCS"].ConnectionString;
SqlCommand cmd = new SqlCommand("Select * from Emp", con);
con.Open();
SqlDataReader dr = cmd.ExecuteReader();
string str = "";
while(dr.Read())
{
   str += dr[0] + " ";
}
ViewBag.Result = str;
con.Close();
```

In Index.cshtml, write as below:

All EmpIds = @ViewBag.Result

In web.release.config and web.debug.config provide connectionStrings section as below

```
<connectionStrings>
    <add name="DemoDbCS"
     connectionString="<Value copied in notepad>"
     xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
</connectionStrings>
```

Step 4: Publish and Run the application to note that's Azure Web Application is using SQL Database also in Azure.

Planning the deployment of an Azure SQL Database

- 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.
- In SQL Database, 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.

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	The go-to option for cloud applications with low to medium IO performance requirements, supporting **multiple concurrent queries**. Examples include workgroup or web applications.
Premium	Designed for high transactional volume with high IO performance requirements, supporting many concurrent users. Examples are databases supporting mission critical applications.

To decide on a service tier, start by determining the minimum database features that you need:

Service tier features	Basic	Standard	Premium

DTU	5	10-100	125-4000
Maximum individual database size	2 GB	250 GB	4 TB*
Max in-memory OLTP storage	N/A	N/A	1 – 32 GB
Database backup retention period (Point-In-time restore)	7 days	35 days	35 days

*Subject to availability

DTU Calculator

https://dtucalculator.azurewebsites.net/

Important decision you will need to make is to determine which method you will use to allocate resources across instances of Azure SQL Database.

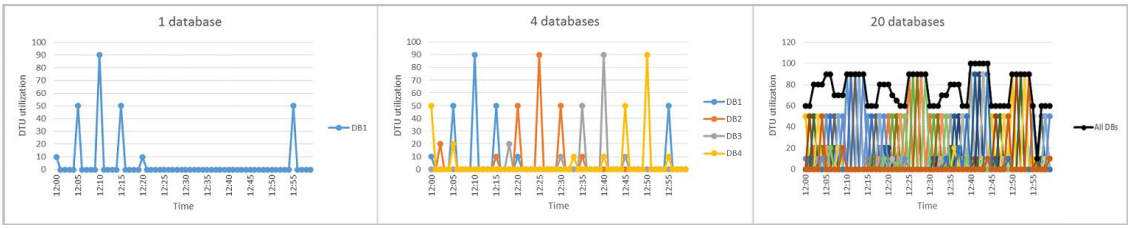
In general, you have two choices, including:

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.**

The following rules of thumb related to database count and database utilization help to ensure that a pool delivers reduced cost compared to using performance levels for single databases.

1. **Minimum number of databases:** If the sum of the DTUs of performance levels for single databases is more than 1.5x the eDTUs needed for the pool, then an elastic pool is more cost effective.
   Example: At least two S3 databases or at least 15 S0 databases are needed for a 100 eDTU pool to be more cost-effective than using performance levels for single databases.
2. **Maximum number of concurrently peaking databases**: By sharing eDTUs, not all databases in a pool can simultaneously use eDTUs up to the limit available when using performance levels for single databases. The fewer databases that concurrently peak, the lower the pool eDTU can be set and the more cost-effective the pool becomes. In general, not more than 2/3 (or 67%) of the databases in the pool should simultaneously peak to their eDTU limit.
   Example: To reduce costs for three S3 databases in a 200 eDTU pool, at most two of these databases can simultaneously peak in their utilization. Otherwise, if more than two of these four S3 databases simultaneously peak, the pool would have to be sized to more than 200 eDTUs. If the pool is resized to more than 200 eDTUs, more S3 databases would need to be added to the pool to keep costs lower than performance levels for single databases.
3. DTU utilization per database: A large difference between the peak and average utilization of a database indicates prolonged periods of low utilization and short periods of high utilization. This utilization pattern is ideal for sharing resources across databases. A database should be considered for a pool when its peak utilization is about **1.5 times greater than its average utilization.**

Example: An S3 database that peaks to 100 DTUs and on average uses 67 DTUs or less is a good candidate for sharing eDTUs in a pool. Alternatively, an S1 database that peaks to 20 DTUs and on average uses 13 DTUs or less is a good candidate for a pool.

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.

In cases where you **can't use tooling**, the following step-by-step can help you estimate whether a pool is more cost-effective than single databases:

1. Estimate the eDTUs needed for the pool as follows:
    a. MAX (<*Total number of DBs* X *average DTU utilization per DB*>,
    < *Number of concurrently peaking DBs* X *Peak DTU utilization per DB*)
    b. 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. For pool storage limits based on eDTU pool size, see eDTU and storage limits for elastic pools and elastic databases.
2. Take the larger of the eDTU estimates from Step 1.
3. See the SQL Database pricing page and find the smallest eDTU pool size that is greater than the estimate from Step 2.
4. Compare the pool price from Step 3 to the price of using the appropriate performance levels for single databases.

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.)

Important:

After you choose the pricing tier and commit your changes by clicking **OK** in the last step, you won't be able to change the pricing tier of the pool.

To change the pricing tier for an existing elastic pool create a new elastic pool in the desired pricing tier and migrate the elastic databases to this new pool.

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.

Elastic Pool Limits:

Service tier features	Basic	Standard	Premium	Premium RS
Pool size (eDTUs / Pool)	50-1600	50-3000	125-4000	125-1000
Max Database Size	Up to 2GB	Up to 250 GB	Up to 500 GB	Up to 500 GB
Maximum data storage per pool	Up to 156 GB	Up to 3000 GB	Up to 750 GB	Up to 750 GB
Max Number of database per pool	Up to 500	Up to 500	Up to 100	Up to 100
Max eDTU per database	5	100	4000	1000

Individual Database Limits:

Service tier features	Basic	Standard	Premium	Premium RS
DTU	5	10-100	125-4000	125-1000
Maximum individual database size	2 GB	250 GB	4 TB*	500 GB
Max in-memory OLTP storage	N/A	N/A	1 – 32 GB	1 – 8GB
Database backup retention period (Point-In-time restore)	7 days	35 days	35 days	35 days

Monitoring Azure SQL Database and Alerting

Monitoring the performance of a SQL database in Azure starts with monitoring the resource utilization relative to the level of database performance you choose. Monitoring helps you determine whether your database has excess capacity or is having trouble because resources are maxed out, and then decide whether it's time to adjust the performance level and service tier of your database.

To Edit Monitoring Chart

1. **Azure Portal → Sql Database → Select the database (demodb)**

2. On the **demodb** blade, note the charts displayed in the **Monitoring** section, which show resource utilization in terms of DTU percentage.
3. Edit the Chart and check "Database size percentage", CPU percentage, DTU percentage, etc… → OK
4. Click on chart and display the **Metrics** blade

Note: *Only metrics with the* ***same unit of measure*** *can be displayed in the chart at the same time. For example, if you select "eDTU percentage" then you will only be able to select other metrics with percentage as the unit of measure*

Adding Alerts:

5. Add Alert with following settings and save (Select Database → Settings → Alert rules (Monitoring)
    - Resource: leave the default setting in place
    - Name: **demodb storage alert**
    - Description: **size alert for demodb database**
    - Metric: **Database size percentage**
    - Condition: **greater than**
    - Threshold: **50**
    - Period: **over the last 5 minutes**
    - Email owners, contributors, and readers: **selected**
    - Additional administrator email(s): any email address
    - Webhook: leave blank

Elastic Pool monitoring

You can go to a particular pool to see its resource utilization. By default, the pool is configured to show storage and eDTU usage for the last hour. The chart can be configured to show different metrics over various time windows.

1. Select a pool to work with.
2. Under **Elastic Pool Monitoring** is a chart labeled **Resource utilization**. Click the chart.
3. On the metric blade, click **Edit**.
4. In the **Edit Chart** blade, select a new time range (past hour, today, or past week), or click **custom** to select any date range in the last two weeks. Select the chart type (bar or line), then select the resources to monitor.

Elastic database monitoring:

Individual databases can also be monitored for potential trouble.

1. Under **Elastic Database Monitoring**, there is a chart that displays metrics for five databases. By default, the chart displays the top 5 databases in the pool by average eDTU usage in the past hour. Click the chart.

2. The **Database Resource Utilization** blade appears. This provides a detailed view of the database usage in the pool. Using the grid in the lower part of the blade, you can select any database(s) in the pool to display its usage in the chart (**up to 5 databases**). You can also customize the metrics and time window displayed in the chart by clicking **Edit chart**.

Configure Azure SQL Database Auditing

Azure SQL Database Auditing tracks database events and writes them to an audit log in your Azure Storage account.

Auditing can help you maintain regulatory compliance, understand database activity, and gain insight into discrepancies and anomalies that could indicate business concerns or suspected security violations.

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 preconfigured reports and a dashboard to get started quickly with activity and event reporting.
- **Analyze** reports. You can find suspicious events, unusual activity, and trends.

There are two **Auditing types**:

1. **Blob auditing** - logs are written to Azure Blob Storage. This is a newer auditing method, which provides **higher performance**, supports **higher granularity object-level auditing**, and is **more cost effective**.

1. **Create a Storage Account**
    a. Name: a valid, unique name for a new storage account
    b. Deployment model: **Classic**
    c. Performance: **Standard**
    d. Replication: **Locally-redundant storage (LRS)**
    e. Subscription: your Azure subscription
    f. Resource Group: **DemoRG**
    g. Location: the same location where you created your Azure SQL Database server
    h. Pin to dashboard: selected
2. Navigate to DemoDb (database) → Settings blade → **Auditing & Threat detection**
3. **Clear Inherit settings from server** check box and apply the following settings:
    a. Auditing: **ON**
    b. Auditing Type: **Storage**
    c. Storage Details: leave the default (pointing to the storage account you created earlier)

d. Audited Events: **All**

e. Threat detection (preview): **OFF**

4. On Auditing Blade select **View audit logs** to point out where you would see audit records.
5. Build a client application with following connection string

```
Server=tcp:server_name.database.windows.net,1433;Database=demodb; User
ID=dsstest;Password=test; TrustServerCertificate=False;Connection Timeout=30;
```

Note: **secure** in front of **.windows.net**

6. Perform CRUD Operations
7. Select Database → **Auditing & Threat detection → View Audit logs.**
8. Note that the **Auditing** blade contains additional **Login** and **DataAccess** events.

Export and Import of Database

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.

Copy Database:

Azure Portal → SQL databases → Select the Database → Click Copy in database blade → Provide the required details → OK.

- Can be either of same or different server
- Service Tier can be changed.

Using PowerShell for Continuous Copy:

This command schedules a continuous copy of the database named Orders on the server named SourceDbServer. The command creates a target database on the server named DestDbServer.

```
Start-AzureSqlDatabaseCopy -ServerName "SourceDbServer" -DatabaseName "SourceDb" -PartnerServer
"DestDbServer" -ContinuousCopy
```

Note: For a continuous copy, the source and target databases **cannot** reside on the same server, and the servers that host the source and target databases **must be part** of the same subscription.

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 the Database
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**:**

Powershell to Export Database:

```
$subscriptionId = "YOUR AZURE SUBSCRIPTION ID"

Login-AzureRmAccount
Set-AzureRmContext -SubscriptionId $subscriptionId

# Database to export
$DatabaseName = "DATABASE-NAME"
$ResourceGroupName = "RESOURCE-GROUP-NAME"
$ServerName = "SERVER-NAME"
$serverAdmin = "ADMIN-NAME"
$serverPassword = "ADMIN-PASSWORD"
$securePassword = ConvertTo-SecureString -String $serverPassword -AsPlainText -Force
$creds = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $serverAdmin,
$securePassword
```

```
# Generate a unique filename for the BACPAC
$bacpacFilename = $DatabaseName + (Get-Date).ToString("yyyyMMddHHmm") + ".bacpac"

# Storage account info for the BACPAC
$BaseStorageUri = "https://STORAGE-NAME.blob.core.windows.net/BLOB-CONTAINER-NAME/"
$BacpacUri = $BaseStorageUri + $bacpacFilename
$StorageKeytype = "StorageAccessKey"
$StorageKey = "YOUR STORAGE KEY"

$exportRequest = New-AzureRmSqlDatabaseExport -ResourceGroupName $ResourceGroupName -ServerName
$ServerName  -DatabaseName $DatabaseName -StorageKeytype $StorageKeytype -StorageKey $StorageKey -
StorageUri $BacpacUri -AdministratorLogin $creds.UserName -AdministratorLoginPassword $creds.Password

# Check status of the export
Get-AzureRmSqlDatabaseImportExportStatus -OperationStatusLink $exportRequest.OperationStatusLink
```

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.

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.

PowerShell:

```
$importRequest = New-AzureRmSqlDatabaseImport -ResourceGroupName "myResourceGroup" -ServerName
$servername -DatabaseName "MyImportSample" -DatabaseMaxSizeBytes "262144000" -StorageKeyType
"StorageAccessKey" -StorageKey $(Get-AzureRmStorageAccountKey -ResourceGroupName "myResourceGroup" -
StorageAccountName $storageaccountname).Value[0] -StorageUri
"http://$storageaccountname.blob.core.windows.net/importsample/sample.bacpac" -Edition "Standard" -
```

```
ServiceObjectiveName "P6" -AdministratorLogin "ServerAdmin" -AdministratorLoginPassword $(ConvertTo-SecureString -String "ASecureP@assw0rd" -AsPlainText -Force)
```

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

Business Continuity

- SQL Database **automatically** creates a database backups and uses Azure read-access geo-redundant storage (RA-GRS) to provide **geo-redundancy**.
- SQL Database automatically performs a combination of **full database backups weekly**, **differential database backups hourly,** and **transaction log backups every five minutes** to protect your business from data loss.
- The full and differential database backups are also replicated to a paired data center for protection against a data center outage.
- These backups are created automatically. SQL Database provides up to 200% of your maximum provisioned database storage as backup storage at no additional cost. If your database exceeds the provided backup storage, you can choose to **reduce the retention period** by contacting Azure Support. Another option is to pay for extra backup storage that is billed at the standard Read-Access Geographically Redundant Storage (RA-GRS) rate.
- Each SQL Database backup has a retention period that is based on the service-tier of the database. The retention period for a database in the:
    - Basic service tier is 7 days.
    - Standard service tier is 35 days.
    - Premium service tier is 35 days.
- **Use automated backups as your business continuity and recovery mechanism ONLY if your application:**
    - Is not considered mission critical.
    - Doesn't have a binding SLA therefore the downtime of 24 hours or longer will not result in financial liability.
    - Has a low rate of data change (low transactions per hour) and losing up to an hour of change is an acceptable data loss.
    - Is cost sensitive.

Recover an Azure SQL Database:

SQL Database provides three options for database recovery using the automated database backups.

1. A new database on the **same logical server** recovered to a specified **point in time** within the retention period.

2. A new database on **any logical server** in any region recovered to the point of the most recent daily **backups in geo-replicated** blob storage (RA-GRS).
3. A database on the **same logical server** recovered to the deletion time for a **deleted database**.

1. Point-in-time restore to same Logical Server:

You can use the **automated backups** to recover a copy of your database to a known good point in time, provided that time is within the database retention period.

After the database is restored, you can either replace the original database with the restored database or copy the needed data from the restored data into the original database.

a) **Azure Portal → SQL databases** → Select database you want to restore → At the top of your database's blade, select **Restore**

b) Provide the required details specially the Restore Point → OK

Using PowerShell:

```
$Database = Get-AzureRmSqlDatabase -ResourceGroupName "DemoRG" -ServerName "Server01" -DatabaseName "Database01"

Restore-AzureRmSqlDatabase -FromPointInTimeBackup -PointInTime UTCDateTime -ResourceGroupName $Database.ResourceGroupName -ServerName $Database.ServerName -TargetDatabaseName "RestoredDatabase" -ResourceId $Database.ResourceID -Edition "Standard" -ServiceObjectiveName "S2"
```

Note: We can restore from "Long Term Backup" also.

Provided we have configured for the same: Database Server → Manage Backup → Configure Long term restore settings.

2. Restore Database to same/different Logical Server (Geo Restore)

If your application's downtime does not result in business liability you can use Geo-Restore as a method to recover your application database(s). It creates a copy of the database from its latest geo-redundant backup.

i. Azure Portal → SQL Databases → +Add
ii. Provide required details specially
    a. Select Source = "**Backup**"
    b. Backup = Database name to duplicate
    c. Server = Replicated Server
iii. Create

Using PowerShell:

```
$GeoBackup = Get-AzureRmSqlDatabaseGeoBackup -ResourceGroupName "ResourceGroup01" -ServerName "Server01" -DatabaseName "Database01"

Restore-AzureRmSqlDatabase -FromGeoBackup -ResourceGroupName "TargetResourceGroup" -ServerName "TargetServer" -TargetDatabaseName "RestoredDatabase" -ResourceId $GeoBackup.ResourceID -Edition "Standard" -RequestedServiceObjectiveName "S2"
```

3. Restore a deleted database to same logical server

If the database is deleted but the logical server has not been deleted, you can restore the deleted database to the point at which it was deleted. This restores a database backup to the same logical SQL server from which it was deleted. You can restore it using the original name or provide a new name or the restored database.

Azure Portal → SQL Servers → Select the Logical Server → In the Summary Blade scroll and go to Operations → Deleted Databases → Select the database to restore

```
$DeletedDatabase = Get-AzureRmSqlDeletedDatabaseBackup -ResourceGroupName "ResourceGroup01" -ServerName "Server01" -DatabaseName "Database01"

Restore-AzureRmSqlDatabase -FromDeletedDatabaseBackup -DeletionDate $DeletedDatabase.DeletionDate -ResourceGroupName $DeletedDatabase.ResourceGroupName -ServerName $DeletedDatabase.ServerName -TargetDatabaseName "RestoredDatabase" -ResourceId $DeletedDatabase.ResourceID -Edition "Standard" -ServiceObjectiveName "S2"
```

Active Geo-Replication

- Active Geo-Replication enables you to configure up to **four readable secondary databases** in the same or different data center locations (regions). Secondary databases are available for **querying and for failover** in the case of a data center outage or the inability to connect to the primary database.
- If the primary database goes offline unexpectedly or you need to take it offline for maintenance activities, you can quickly promote a secondary to become the primary **(also called a failover)** and configure applications to connect to the newly promoted primary. With a planned failover, there is no data loss. With an unplanned failover, there may be some small amount of data loss for very recent transactions due to the nature of asynchronous replication. After a failover, you can later failback - either according to a plan or when the data center comes back online. In all cases, users experience a small amount of downtime and need to reconnect.
- It is used to reduce recovery time and limit data loss associated with a recovery:

- The secondary database must be in the **same service tier** as the primary, so migrating your primary database to a different service tier requires you to either terminate the geo-replication link and rename the secondary database, or simply drop it.

Active Geo-Replication capabilities

1. Automatic Asynchronous Replication
2. Multiple Secondary databases.
3. Readable secondary databases.
4. Active geo-replication of elastic pool database.
5. Secondary database can have lower performance level (DTU) than primary.
6. User-controlled failover and failback.
7. Keeping credentials and firewall rules in sync

Use Active Geo-Replication if your application meets any of these criteria:

- Is mission critical?
- Has a service level agreement (SLA) that does not allow for 24 hours or more of downtime.
- Downtime will result in financial liability.
- Has a high rate of data change is high and losing an hour of data is not acceptable?
- The additional cost of active geo-replication is lower than the potential financial liability and associated loss of business.
- Key benefit is that the secondary databases are readable and can be used to offload read-only workloads such as reporting jobs.
- **Database migration**: You can use Active Geo-Replication to migrate a database from one server to another online with minimum downtime.
- **Application upgrades**: You can create an extra secondary as a fail back copy during application upgrades.

Enable Geo Replication Backup:

1. Azure Portal → SQL databases → Select database → Scroll database blade → Configure Geo Replication
2. Select Target Region → Provide required details → OK

Note: The non-readable secondary type will be retired and existing non-readable databases will automatically be upgraded to readable secondaries.

To failover to a secondary and promote as primary:

1. Primary Database → On the SQL Database blade, select All settings → Geo-Replication.
2. In the **SECONDARIES** list, select the database you want to become the new primary and click **Failover**.

Note: There is a short period during which both databases are unavailable (on the order of 0 to 25 seconds) while the roles are switched. If the primary database has multiple secondary databases, the command automatically reconfigures the other secondaries to connect to the new primary. The entire operation should take less than a minute to complete under normal circumstances

Powershell:

Cmdlet	Description
Get-AzureRmSqlDatabase	Gets one or more databases.
New-AzureRmSqlDatabaseSecondary	Creates a secondary database for an existing database and starts data replication.
Set-AzureRmSqlDatabaseSecondary	Switches a secondary database to be primary to initiate failover.
Remove-AzureRmSqlDatabaseSecondary	Terminates data replication between a SQL Database and the specified secondary database.
Get-AzureRmSqlDatabaseReplicationLink	Gets the geo-replication links between an Azure SQL Database and a resource group or SQL Server.

Summary / Comparison table

The following table compares the ERT and RPO for the three most common scenarios.

Capability	Basic tier	Standard tier	Premium tier
Point in Time Restore from backup	Any restore point within 7 days	Any restore point within 35 days	Any restore point within 35 days
Geo-Restore from geo-replicated backups	ERT < 12h, RPO < 1h	ERT < 12h, RPO < 1h	ERT < 12h, RPO < 1h
Active Geo-Replication	ERT < 30s, RPO < 5s	ERT < 30s, RPO < 5s	ERT < 30s, RPO < 5s
Restore from Long Term Backup	ERT < 12h, RPO < 1 wk	ERT < 12h, RPO < 1 wk	ERT < 12h, RPO < 1 wk

ERT = Estimated Recovery Time

RPO = Recovery Point Objective is the maximum amount of recent data updates (time interval) the application can tolerate losing when recovering after the disruptive event.

New Feature:

https://docs.microsoft.com/en-us/azure/sql-database/sql-database-get-started-sql-data-sync

Long-Term Backup Retention

The **Long-Term Backup Retention** feature enables you to store your Azure SQL Database backups in an **Azure Recovery Services vault** for up to 10 years. You can store up to 1000 databases per vault. You can select any backup in the vault to restore it as a new database.

To configure long-term backup retention for a database:

1. Create an Azure Recovery Services vault in the **same region, subscription, and resource group** as your SQL Database server.
2. Register the server to the vault
3. Create an Azure Recovery Services Protection Policy
4. Apply the protection policy to the databases that require long-term backup retention

To recover from a long-term backup retention backup:

1. List the vault where the backup is stored
2. List the container that is mapped to your logical server
3. List the data source within the vault that is mapped to your database
4. List the recovery points available to restore
5. Restore from the recovery point to the target server within your subscription

Support for AccessToken in SqlConnection

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.

The following sample shows how to authenticate SQL Server connections using Azure Active directory:

```
// get access token using ADAL.NET
var authContext = new AuthenticationContext(authority);
var authResult = await authContext.AcquireTokenAsync(appUri, clientCredential);
// setup connection to SQL Server
var sqlConnection = new SqlConnection(connectionString);
sqlConnection.AccessToken = authResult.AccessToken;
await sqlConnection.OpenAsync();
```