**Implementing a Relational Database in Microsoft Azure SQL Database**

**Course Overview**

**Course Overview**

Azure SQL database is Microsoft's main relational database offering in the cloud.

Major topics:

* Deployment options and purchasing models,
* Point in time and long-term retention backups,
* Scheduled database jobs, and
* Automatically synchronize data within multiple databases.

**Getting Started**

**Overview**

* Database offerings available in Microsoft Azure:
  + Infrastructure as a Service ( IaaS)
  + Platform as a Service, (PaaS)
* Azure SQL Database is a DB- as-a-service offering.
* Following are few imp concepts related to provisioning new instance of Azure SQL DB:
  + Deployment Options: Single database, Elastic pool, or Managed instance deployment options
  + Purchasing Models: Virtual core and DTU based models.
  + Service Tiers: General purpose, Business critical, and Hyper-scale

**IaaS vs. PaaS Azure Database Offerings**

* Before Cloud, to get a SQL server database provisioned we would get the server either on-premises or in your data center and install SQL Server on that box. We were responsible to configure the networking, the security, operating system patching, and the performance and scaling of our SQL Server instance.
* Azure has 2 options for SQL Server based workloads:
  + Azure SQL Database:
    - Database as a Service Offering (Falls into Platform as a Service - PaaS offering)
    - We simply create resource of type Azure SQL Database and create your database and tables in this instance
    - We don’t have to worry about getting a computer, installing operating system, installing the database engine, patching the operating system, and scaling your database.
    - We benefit from features like built-in high availability, intelligence, and database management tools (which are not available on SQL Server)
    - We have option to Scale up or Scale out without any interruption in the service, which is a good option for highly available applications.
    - This option has multiple deployment options, purchasing models, and service tiers.
  + SQL Server installation on VM:
    - IaaS offering.
    - This is a SQL Server inside a fully managed virtual machine in Azure.
    - Choose this option for migrating on-premises SQL Server databases and applications without any database change. (Lift-and-Shift Scenario).
    - The SQL Server installed on a VM is identical to the SQL Server you have installed on-premises.
    - We still don't need to worry about the physical machine. Many aspects of this virtual machine, including the disks, are fully managed by Azure.
    - Since it’s still an instance of VM, we need to manage the instance of SQL server manually.
    - This IaaS option gives us full control over the database engine. We could control the timing of maintenance and patching of your database engine.

**Comparing IaaS and PaaS Offerings:**

* Advantages of SQL Server on VM:

1) 99.95% availability guaranteed by Azure.

2) Full control over SQL engine.

3) Full Parity with the matching version of on-premises SQL Server. This means easy migration from SQL Server on-premises to Azure.

4) SQL Server VM can be placed in a virtual network subnet and assign a private IP address to this VM. This makes SQL Server installation more secure.

* Advantages of Azure SQL DB:

1) 99.99% availability guaranteed by Azure

2) Microsoft Azure takes care of backups, patching, and recovery. We don't need to worry about creating backups, do the patching, and perform recovery manually.

3) We have the ability to assign necessary resources, for example CPU and storage, to individual databases.

4) Built-in advanced intelligence and security.

5) We can scale down, up, in, or out without any downtime.

* Disadvantages of SQL Server on VM:

1) Manually manage backups and patches. No automatic backups

2) We need to implement our own high availability solution. Ex: provisioning multiple virtual machines and place them in an availability set.

3) There is downtime while changing the resources.

* Disadvantages of Azure SQL Databases:

1) Migration from SQL Server on premises may be difficult. There are features, which are not supported in Azure SQL Database

2) Virtual machines are abstracted out from clients. This means you don't have any control over the exact maintenance time for the OS or the SQL engine.

3) We cannot assign private IP addresses to Azure SQL Databases with the exception of managed instances.

* Bottom line is that additional control comes with added responsibility of managing VM and underlying DB engine.

**Azure SQL Database Deployment option:**

* Why business would choose to move to Azure SQL Database?

Answer: Moving to Azure SQL Database reduces the amount of time that you need to invest to administer the database. For most companies moving to a cloud service is about offloading complexity of administration. We can still continue to administer your database, but no need to manage the database engine, OS, or hardware. For example, we manage logins, indexes, and tune your queries. We can manage security and auditing and high availability.

* Azure SQL DB offers 3 deployment models :

1) Single Database: Single isolated database that is perfect for applications that need a single data source. Each database receives its own guaranteed compute, memory, and storage.

2) Elastic Pool: A collection of single databases with a shared set of resources, such as CPU or memory.

There is a fixed amount of resources, which will be shared by all databases in the pool.

3) Managed Instance: A Managed instance refers to a Server. It contains set of DBs that can be used together. Closest deployment option to the IaaS scenario offered by Azure SQL Database. This means easy migration from on-premises databases. Each managed instance has its guaranteed resources.

**Azure SQL Database purchasing Model:**

* Two Purchasing Models:

1) DTU based:

* + A database transaction unit, or DTU, represents a blended measure of CPU, memory, reads, and writes.
  + This guarantees a certain level of compute, storage, and I/O resources.
  + We assign this bundled unit (DTU) to our Azure SQL DB. We cannot adjust individual resources, such as compute or memory.
  + DTU-based purchasing model is only supported by single database and elastic pool.
  + Best for customers who want simple, preconfigured resource options.
  + We might need to calculate the needed amount of DTUs before migration from your on-premises database. There are tools, such as DTU Calculator, which can be used for this purpose.
  + Microsoft recommends Vcore Model. However if the DTU based purchasing model meets perf and business requirements, we are not obligated to change to Vcore Model.

2) V-Core Model:

* + Newer offering comparing to DTU-based. Microsoft recommends vCore-based purchasing model.
  + This purchasing model gives us the option to choose between generations of hardware, number of CPU cores, memory, and storage size. This provided more flexibility in adjusting individual resources.
  + Supported by all three deployment options, including single database, elastic pools, and managed instance. (Remember DTU model is not supported by managed instances)
  + Best for customers who need flexibility, control, and transparency. Ex: We have the flexibility to adjust individual resources, such as CPU, memory, and storage.
  + Gives a straightforward way to translate on- premises workloads to the cloud. We know how much resources we use on our on-premises server. We can easily translate those resources to the corresponding values for the vCore-based model. (Unlike DTU based, where we might have to use DTU calculator)
  + Microsoft recommends VCore model.

* Converting to V-Core model:
  + If single database or elastic pool consumes more than 300 DTUs, converting to the vCore-based model might reduce costs.
  + We can convert to the vCore-based model by using your API of choice, meaning Azure CLI, RESTful API, or PowerShell or by using the Azure portal, and there is no downtime. (biggest upside)
  + Azure SQL Database managed instance only supports vCore-based purchasing model. So if we are going with this deployment option, we have to choose vCore-based purchasing model.

**Azure SQL Database Service tier:**

After selecting deployment options and purchasing models, we have to choose he Service tiers.   
For DTU based model we have service tiers: Basic, Standard, Premium  
For Vcore based model, we have service tiers: General Purpose, Business Critical and hyperScale.

Following are the options

1. General Purpose/ Standard:
   * Designed for most generic workloads. Good enough for most workloads.
   * 99.99% SLA graranteed.
   * Storage speed: 5-ms read latency and 10-ms write latency
   * Max 4TB Databases.
2. Business Critical/Premium:
   * designed for applications requiring low latency.
   * 99.99% SLA guaranteed.
   * Storage speed: Storage is much faster in this service tier. 1-ms read latency and 2-ms of writing latency.
   * Max 4TB Databases.

Both these service tiers are available for all deployment options.

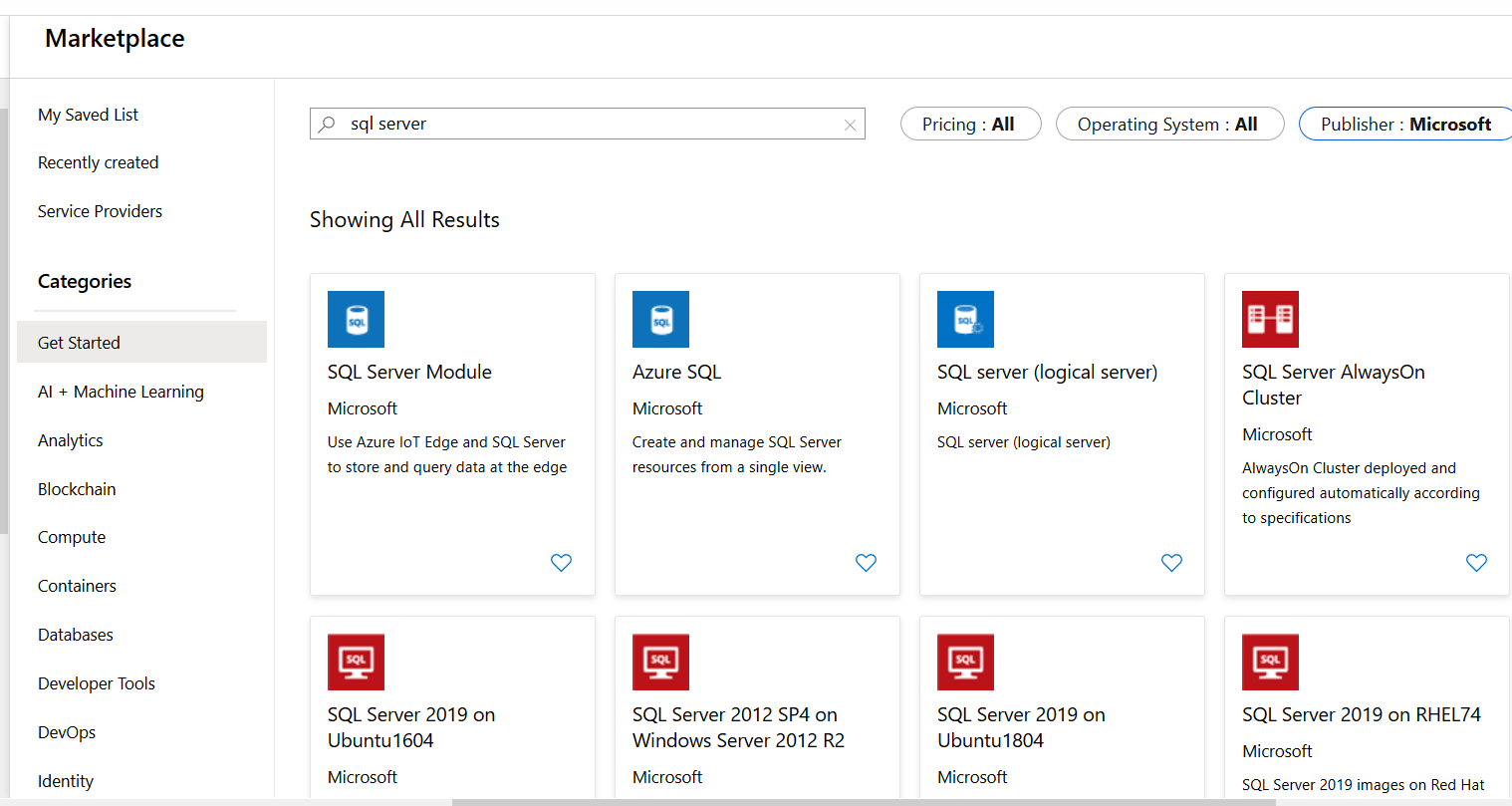
1. HyperScale:
   * intended for customers who have large databases up to 100TB in size.
   * Only available for the vCore purchasing model.

Summary of what we have learnt so far:

* + The first one is the deployment option. You can choose from single database, elastic pool, or managed instance.
  + After choosing the deployment option, you need to choose a purchasing model. You can choose from DTU-based or vCore-based. The DTU-based purchasing model is only available for single database and elastic pool.(.i.e not available for managed instances) The vCore-based purchasing model is available for all deployment options.
  + After decided the purchasing model, you need to decide the service tiers. You can choose among general purpose or standard, business critical or premium, and hyperscale. The hyperscale service tier is only available for the vCore-based purchasing model.

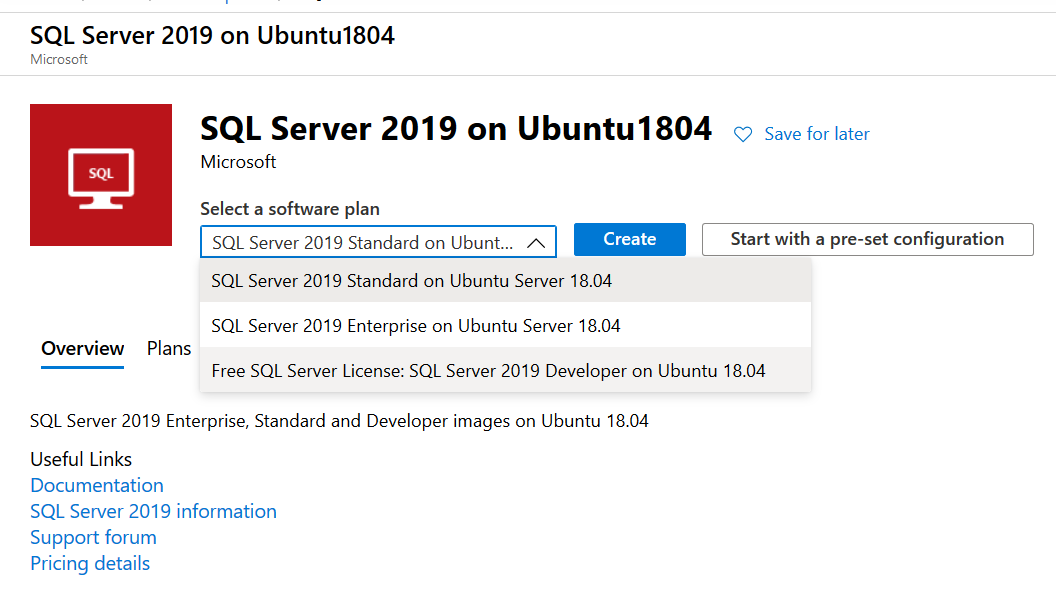
**Demo Provisioning IaaS SQL Server Worload:**

1. Search for Microsoft SQL server images in marketplace.



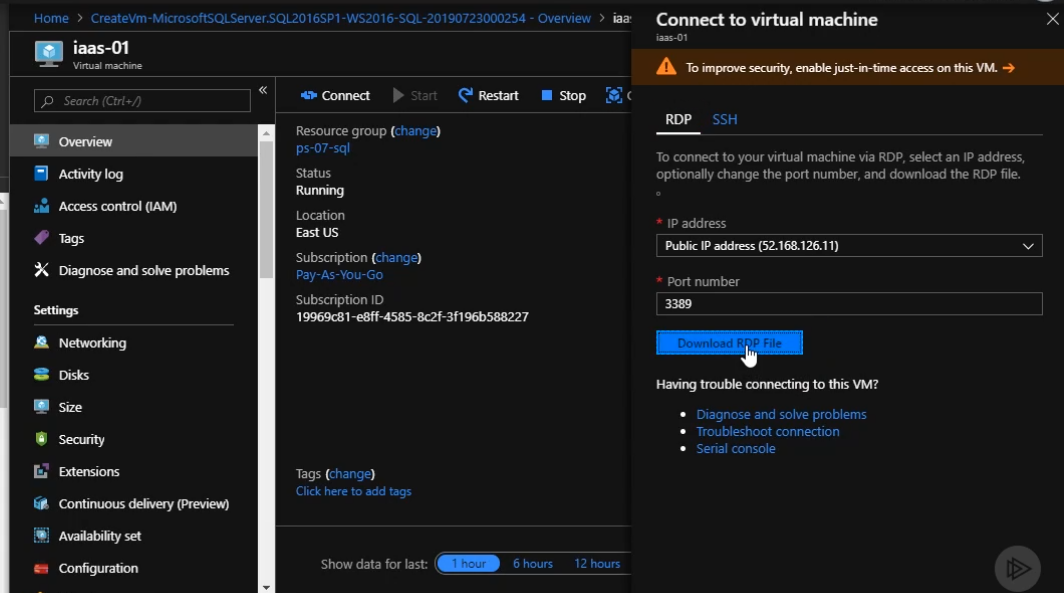
We can choose any one of the images depending upon what we need.

1. Click on one of them and select free liscence



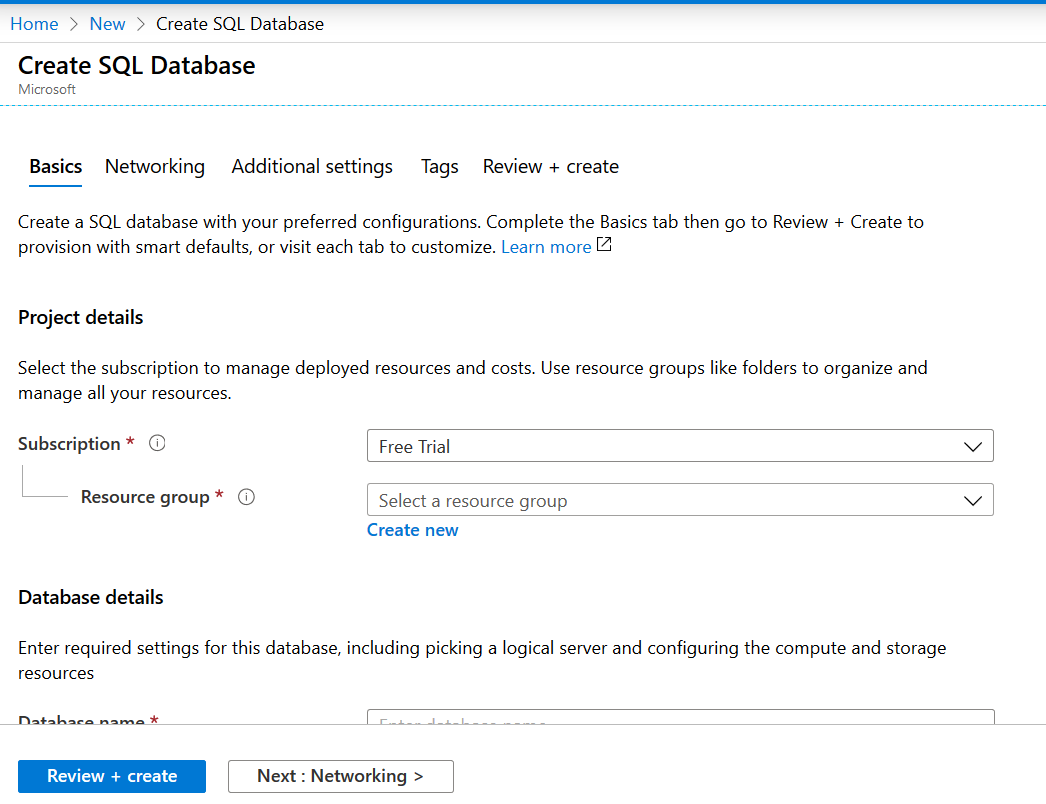
Please note that you still will be charged for the virtual machine and the Windows installed on it. The SQL Server license will be free.

1. Click on Create and configure the VM, Choose the correct size for VM, Admin account, configure inbound ports, OS disk types (Standard/Premium SSD, HDD), Vnet Configuration, SQL Server Connectivity and port, SQL Server Authentication , Tags and finally review and create.



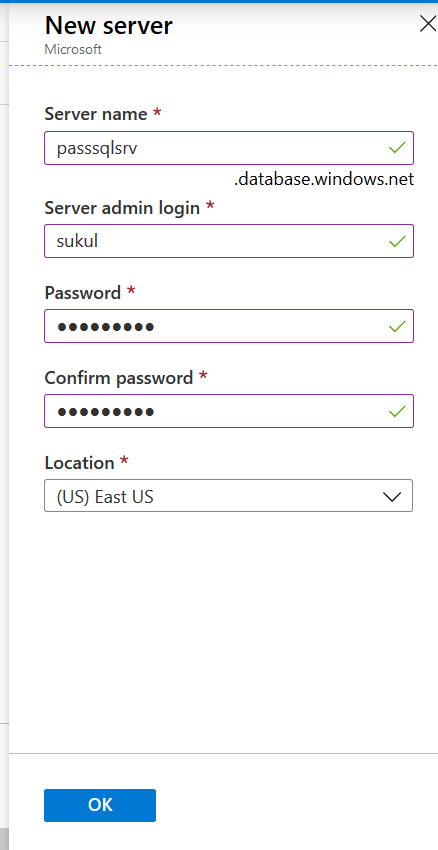
**Demo: Provisioning a PaaS Workload:**

1. Search for SQL Database and create.

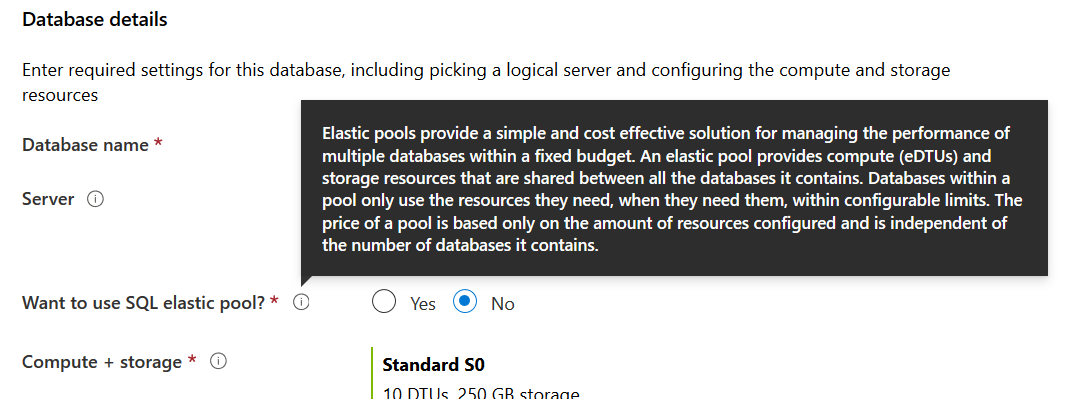


1. Here we provide the resource group, Database Name, Logical server Name. If server does not already exist, we create a new one. Remember that All Azure SQL Database instances are placed inside the logical server.

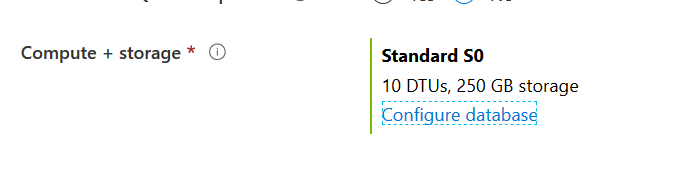
When we choose a to create a new server following are the options available



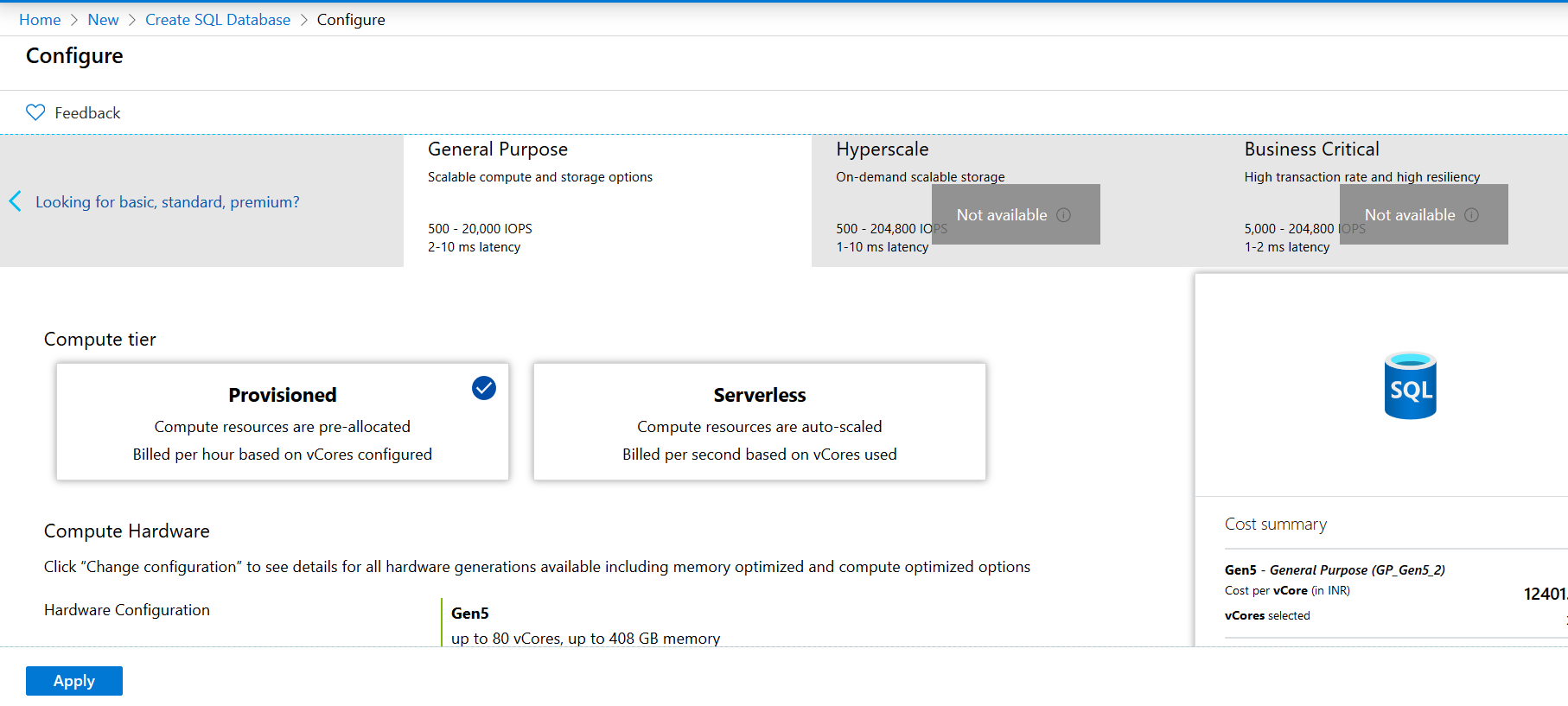
1. Next question we answer is “Want to use SQL elastic pool?”. By answering no we are creating a single Database instance. By choosing Yes we are putting the DB in an elastic pool.

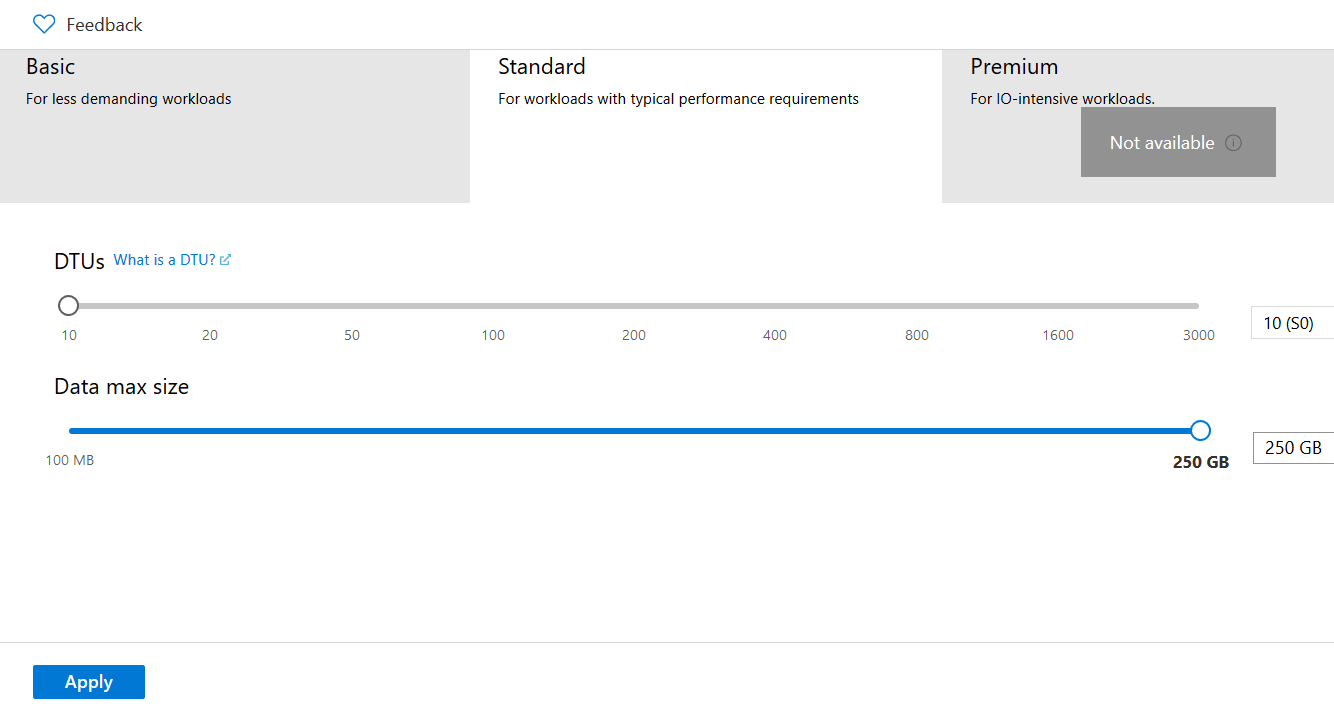


1. Next we click on “Configure Database” to choose the purchasing model and service tier. BY default it allows us to select VCore model. We can choose the DTU based purchasing model as well.

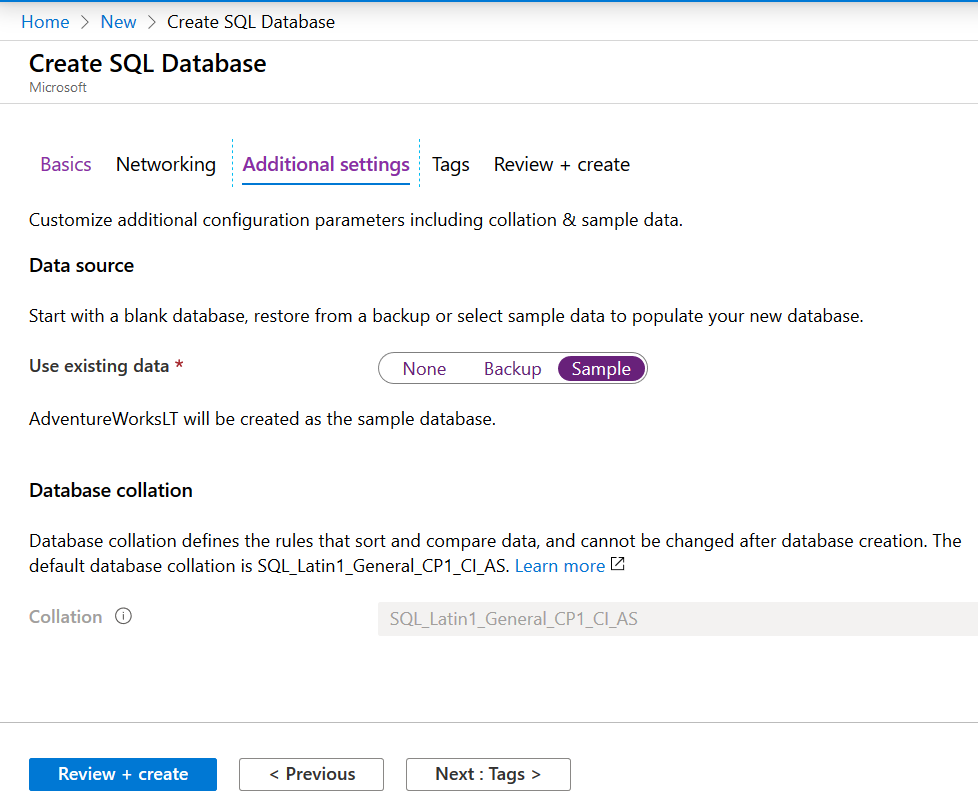


Note that when using free subscription, the default selection is DTU based. Also note that all service tiers are not available.

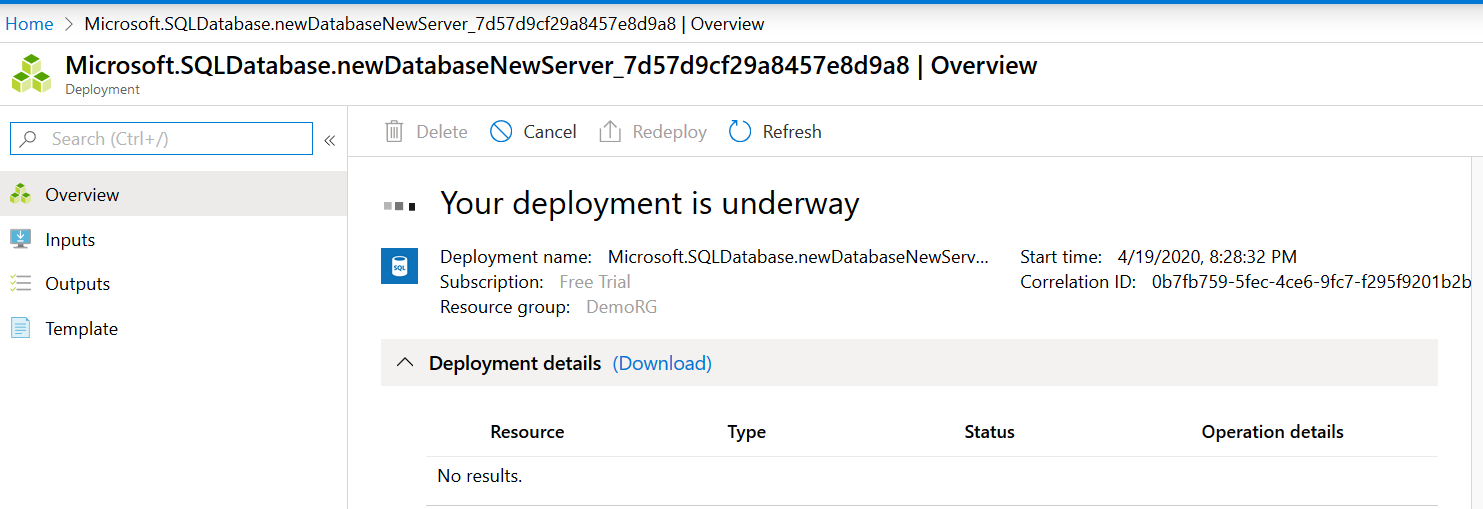




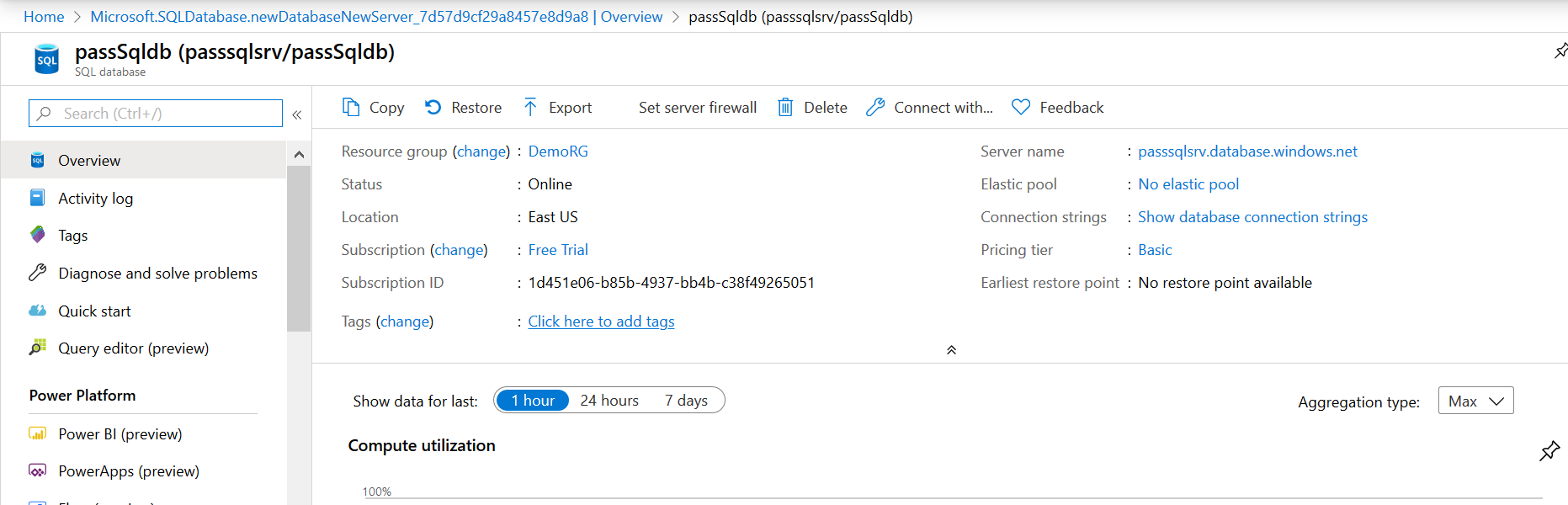
1. Additional settings allow us to create empty db, from a backup or a sample db. We can also specify a collation sequence.



1. Next we specify tags and create the SQL DB instance.



1. Once the deployment completes we can go to the SQL DB Dashboard. We can see that new server has been created.



**Summary:**

* Azure Offering for SQL workload: 1) SQL Server installed on an Azure VM 2) Azure SQL Database or Database-as-a-Service.
* Deployment Options: single database, elastic pool, or managed instance deployment options.
  + The first option gives you a single database with guaranteed dedicated resources.
  + The second option, elastic pools, allows you to bundle a few single databases inside an elastic pool and assign fixed amount of resources to this elastic pool. These single databases have to share these resources among each other.
  + Managed instance is closest Azure SQL Database deployment option to the IaaS scenario.
* Purchasing model: There are two purchasing models to choose from, vCore-based and DTU-based.
  + In the DTU-based purchasing model, we assign a bundle of resources to your Azure SQL Database. We cannot adjust individual resources, such as CPU or memory.
  + In the second purchasing model, vCore-based, we can individually assign and adjust resources to our Azure SQL Database instance.
* Service Tiers: There are three service tiers offered by Azure SQL Database, general purpose, business critical, and hyperscale.
  + Hyperscale is only available for the vCore- based purchasing model.
  + The general purpose service tier can be used for most workloads.
  + The business critical service tier offers much faster storage.

**Configuring SQL Agent Jobs**

**Overview**

* Elastic database jobs are scheduling technologies for Azure SQL Database single instance and pooled databases. We cannot use elastic jobs for Azure SQL Database managed instance. For our scheduling needs in Azure SQL Database managed instance we should use the SQL agent jobs.
* These are the same SQL agent jobs we used back in the days with Microsoft SQL Server.
* SQL agent jobs for Azure SQL Database managed instance has a few limitations compared to the SQL agent on Microsoft SQL Server.

**Understanding SQL Agent jobs for Managed instances**

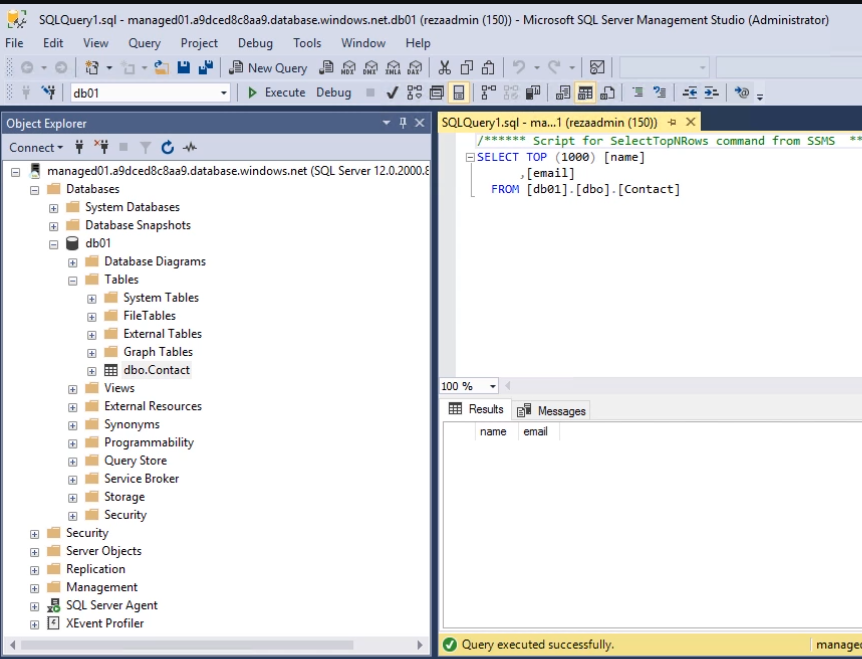
* We have two scheduling technologies native to Azure SQL Database.
  + Elastic Database jobs: technology to create scheduled jobs for single and pooled databases.
  + SQL Agent jobs: used to create scheduled jobs for managed instances.
* SQL agent jobs are series of T-SQL scripts to run against your database. SQL agent jobs can be used to run administrative tasks.
* We can configure these jobs to execute once or multiple times.
* Also these jobs can be monitored for success or failure.
* A job can run on one local server or on multiple remote servers.
* Unlike elastic database job, which is an independent entity in Microsoft Azure, a SQL agent job is an internal database engine component that is executed within the managed instance service.
* SQL agent jobs are not available for single and pooled instances.
* 3 components of SQL Agent job:
  + Job Step: A SQL agent job is a set of one or many steps that should be executed.
  + Schedule: Using the job schedule, we can adjust the timing of your job execution.
  + Notifications: We can define rules that will be used to notify operators, for example administrators, via email once the job completes.(Success or failures)

**Understanding SQL Agent Job Step, Schedule and Notifications:**

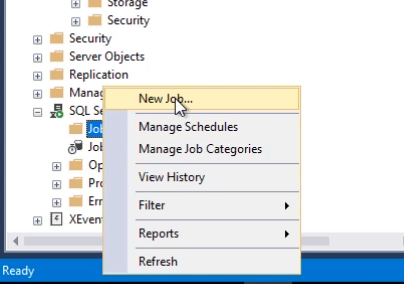
* Job Step:
  + A Job step is a sequence of actions that SQL agent should execute.
  + For each job step, we can define retry strategy and action when the step succeeds or fails.
  + Types of job steps : T-SQL job steps, operating system commands, PowerShell scripts, or SSIS packages, Replication steps( used to publish local db changes to other DB)
* Schedules:
  + Specifies when the job runs
  + Multiple jobs can run on the same schedule. Multiple schedules can apply to the same job
  + The schedules are independent entities.
  + We can also assign multiple schedules to a single job, so this job will be executed each time the criteria of any of the schedules is met.
  + We can also schedule a job to execute whenever the managed instance is restarted.
  + We can run the job one time at a specific date and time or on a recurring schedule.
* Notifications:
  + Managed instance SQL agent jobs enable us to get notifications when the job finishes successfully or fails. So these notifications can be sent in a form of an email.
* SQL agent jobs in managed instances have a few limitations comparing to the SQL agent jobs on Microsoft SQL Server. Following are the differences:
  + SQL agent settings in managed instances are read-only. In Microsoft SQL Server, we could use the stored procedure set\_agent\_properties to overwrite a few of the default properties of an agent. This is not supported in managed instances.
  + In Microsoft SQL Server, we could enable or disable an agent. This feature is not currently supported in managed instances. This means an agent is always running and can't be stopped. So if we don't want a job to get executed, simply remove the schedule from it.
  + In Microsoft SQL Server, you could use emails, pagers, and the NetSend command to send notifications. SQL agent in Azure SQL Database managed instance only supports email notifications. Pages, NetSend, and others are not supported.

**Demo: Creating a SQL Agent Job and Configuring SQL job Email Notifications:**

Below shows a table created in a managed instance DB. The SQL agent job we create will add a row to this table on every execution.

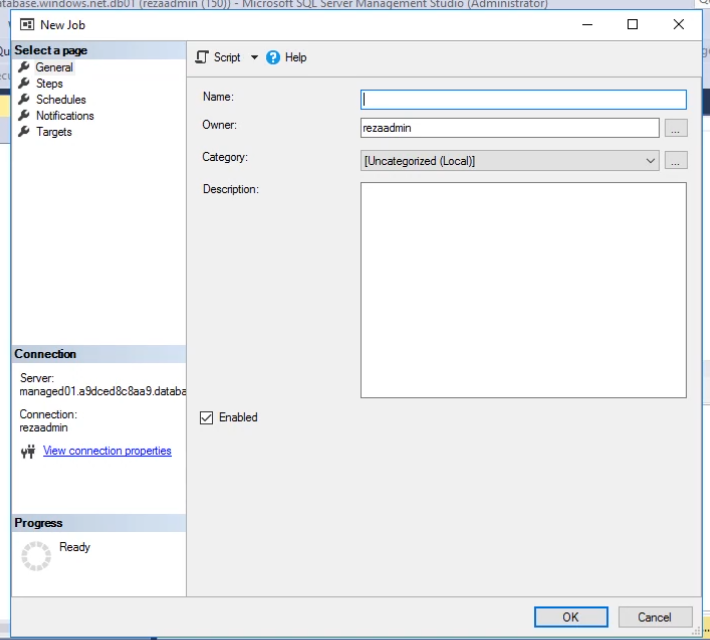


To create job we can make use of TSQL or can use the SSMS graphic user interface.

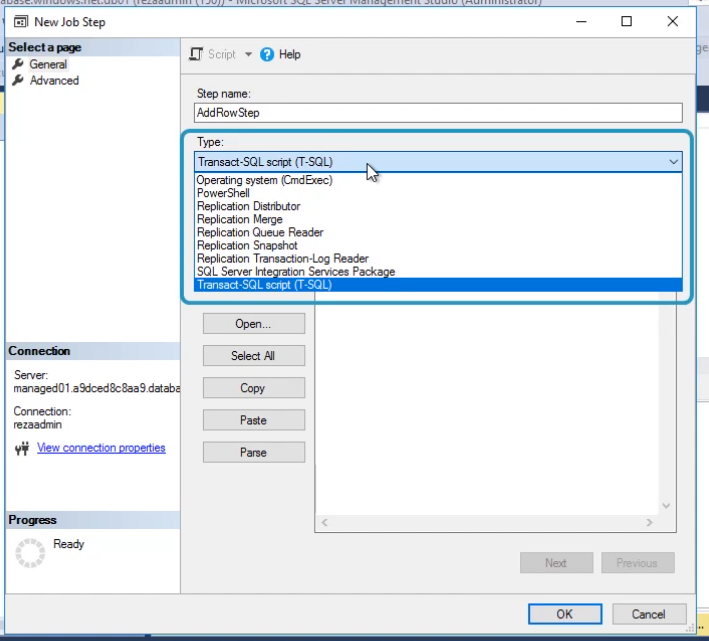
We can create jobs under SQL Server Agent node.

Here on the job definition UI we provide a name for the job.

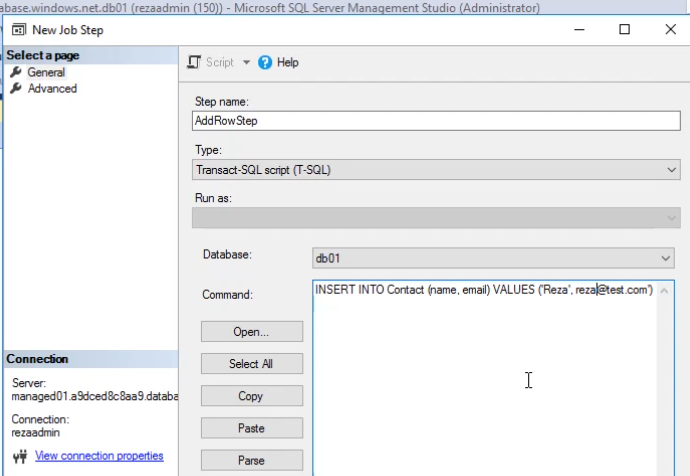
Under Steps we create all the steps in the job.



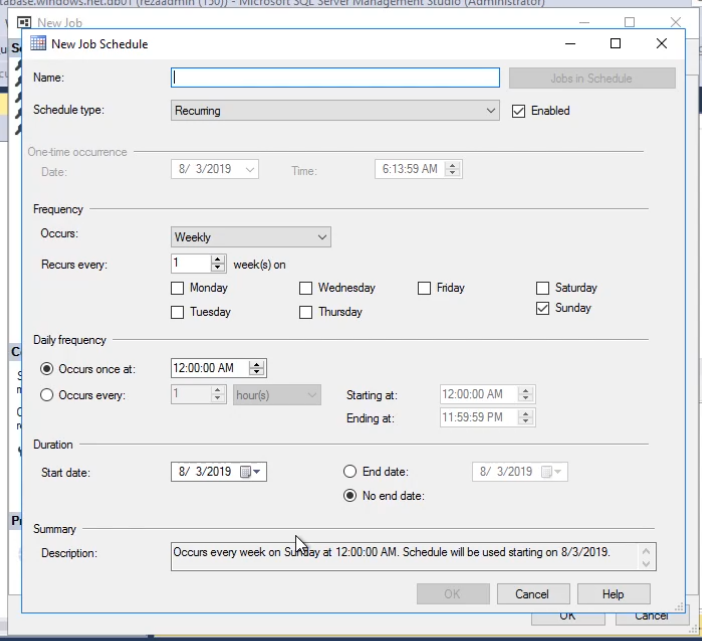
Click New to create a new job step. Note that we have multiple types of steps. Common being TSQL. However we can also use Commandline, Powershell, SSIS packages etc.



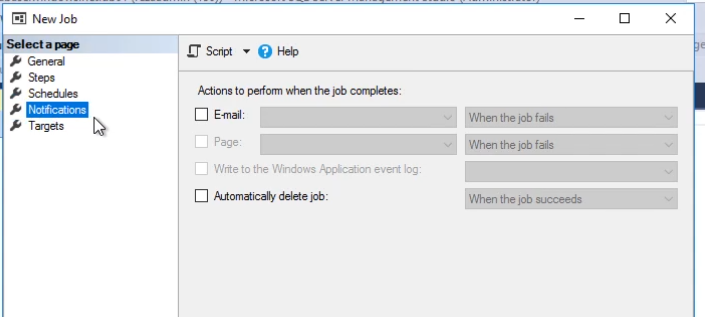
Here we provide the target DB and the command to run. Below runs a normal insert command.



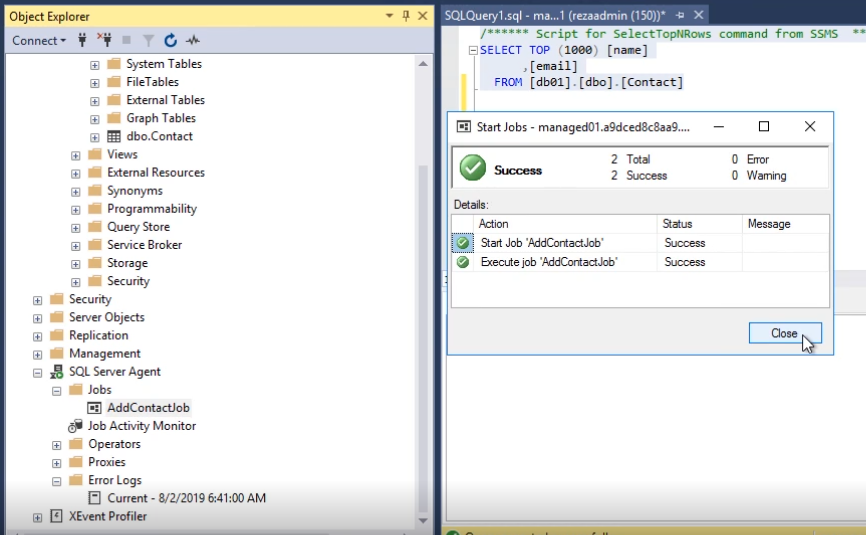
In the next step we specify the job schedule. We can specify a schedule or skip this and run the job manually.



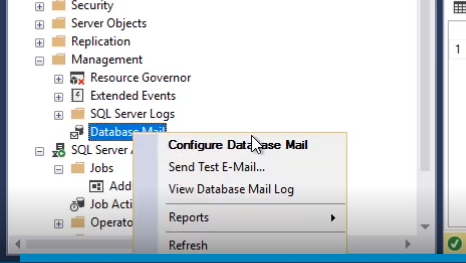
In the Next step we can configure notification. We can send email when job succeeds or fails.



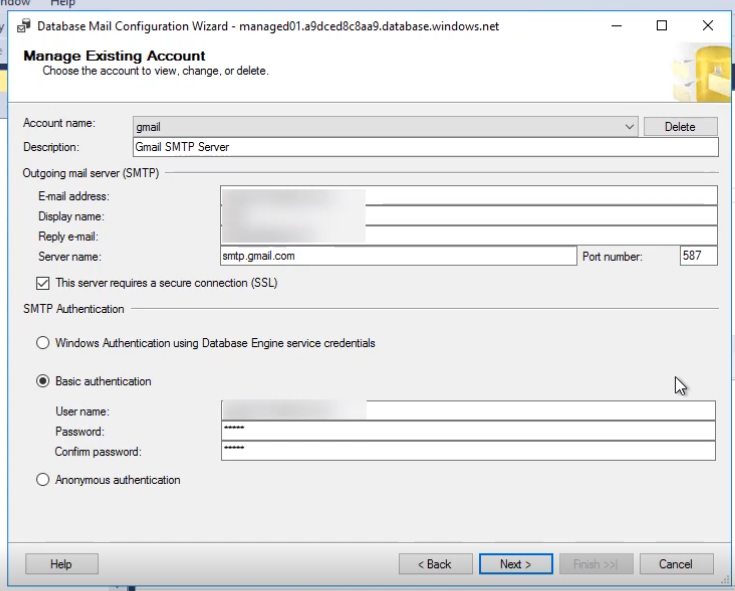
Right click on the job defined and run the job manually. On success we can see the below



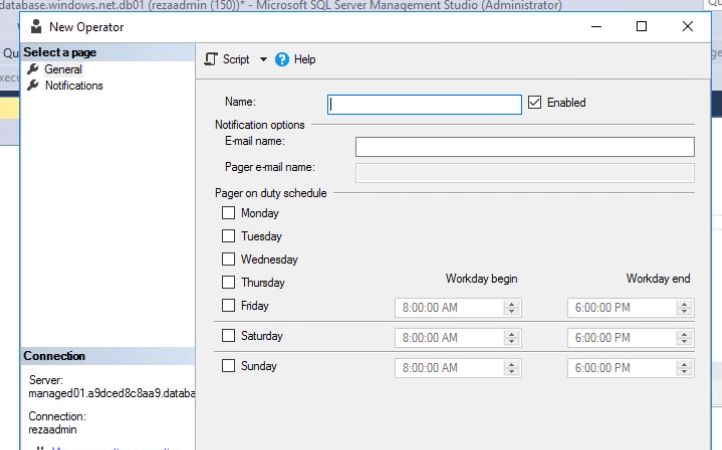
Note that before we configure email notifications for the job, the database email should be configured correctly.



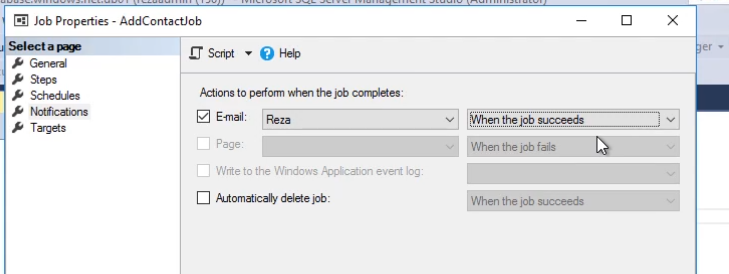
Azure SQL Database or SQL Server cannot send emails on their own. We need a POP or SMTP server to be able to send emails. Below shows the use of Gmail free SMTP server to send an email.



After the Database email is correctly configured and correct NSG rules are put in place, we can create an operator. An operator could be a database administrator who needs to get notified on agent jobs' failures or success. So let's right-click on Operators and click on New Operator.



After the operator is defined, we can go back to job definition and go to notification page to configure email when the job succeeds/fails/completes.



**Summary:**

* Elastic jobs are not available for managed instances, so we have to use the SQL agent jobs.
* Few use cases for SQL agent jobs include running administrative tasks or syncing data between a few databases.
* SQL agent jobs have three main components: The job step, the schedule, and the notification.
* A SQL agent job is a group of job steps that get executed sequentially. We can specify retry logic for each job step. We can also control what to happen if a job succeeds or fails.
* Using a schedule, we can control the timing of your job execution. We can have multiple schedules assigned to a single job or have one schedule assigned to multiple jobs.
* Azure SQL Database managed instance supports email notifications. We can configure your job to send notifications each time the job completes(success, failure, or both)
* SQL Agent on Managed instance Limitations:
  + All properties are read only. The properties of the SQL agent can't be changed with SQL agent jobs for managed instances.
  + Only email notifications are supported for managed instances. We cannot use the NetSend command, pager, or alerts.

**Managing Data Synchronization between Azure and SQL Server On-premises**

**Overview**

* Data Sync is a technology used to sync data between Azure SQL DB as hub Database and any other SQL DB , including on-premises one.
* This Technology is used to sync multiple databases with the same schema. Note that these databases could be hosted in Azure virtual Machines or on-premises.

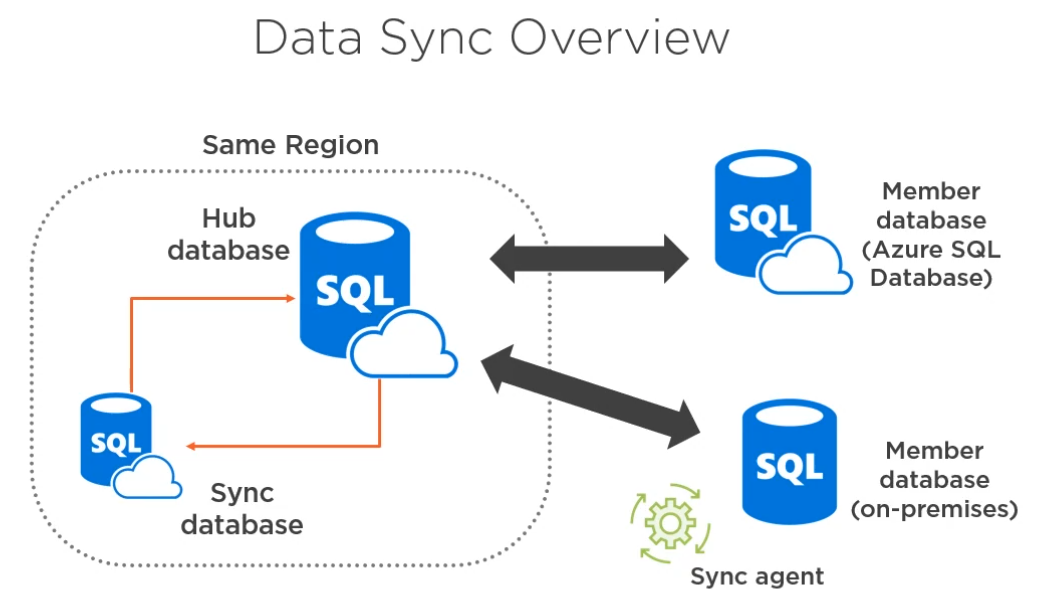
**Azure SQL Data Sync use Cases**

* SQL Data Sync is an Azure SQL Database service that lets us synchronize the data bi-directionally across multiple SQL databases and SQL Server instances.
* Data Sync is useful when data needs to be kept up-to-date across several Azure SQL databases or SQL Server databases.
* Use Cases:
  + Hybrid Data sync: Say we want to sync data between on-premises SQL Server DB and Azure SQL databases.
  + Distributed applications: can have several different workloads across different identical databases. This way you can scale more quickly. Ex: We can use on instance only for reporting.
  + Globally Distributed applications: We can easily sync databases in regions around the world.
* Following is what Data Sync is not:

|  |  |
| --- | --- |
| **What It is not** | **Alternative services** |
| Data Sync is not a Disaster Recovery | Azure geo-redundant backups. |
| If we only need multiple versions of DB and we shall only write to one of them, then we should not use Data Sync | For such scenarios, we can use read-only replicas. |
| Data Sync is not a best tool for ETL scenarios | Use SSIS or data factory. |
| Azure SQL Data Sync is not a migration tool and shouldn't be used to migrate data from on-premises SQL Server to Azure SQL Databases | For such use cases we should use Azure Database Migration Service to migrate data from on-premises SQL Server to Azure SQL Database |

**Sync Group, Sync Group Properties, Hub, Member, and Sync Databases**

* Sync Group:
  + Group of databases that we want to synchronize.
  + Among these databases, we select one database as the hub database. The rest of the databases are member databases.
  + The synchronization occurs only between the hub and individual members. This means if we have two member databases and we change data in one of the member databases, the update doesn't happen directly between member 1 and member 2. It happens through the hub database.
  + The sync group has some properties that we need to configure:
    - Sync Schema: Describes which data is being synchronized. We can select multiple tables or even drilldown to specific columns within each table.
    - Sync Direction: The sync direction can be bi-directional or can flow in only one direction. If I go with the bi-directional option, any update I make in the hub database will reflect in member database and vice versa. In the one-way scenario, the update happens only to the hub or the member databases.
    - Sync Interval: We can trigger the data sync process manually or automatically. To be able to run data sync automatically, we need to set the sync interval. The sync interval describes how often synchronization occurs. Ex: Every hour.
    - Conflict resolution policy: Specifies what should be done in case a conflict happens between hub database and member databases. And we have two options. We can make the hub data overwrite the changes in the member database or vice versa. The first option is called hub wins and the second one member wins.



* Architecture:
  + We have a Hub Database. This is a single database and needs to be an Azure SQL Database. We cannot have a SQL Server on-premises as a hub database.
  + We can have multiple member databases.
  + There is another database called sync database, which is created automatically by Azure Data Sync. The sync database keeps the metadata for our sync job, for ex: the sync groups, the frequency of the sync, and properties
  + The hub database and the sync database should live in the same region. But the member databases are not bound to the same limitation.
  + To be able to use SQL Server on- premises as our member database, we need to manually install and configure a sync agent. This is not the case for Azure SQL Databases or SQL database virtual machines provisioned in Azure.
  + Also the hub and sync databases must be Azure SQL Databases.

**Data Sync considerations and Limitations:**

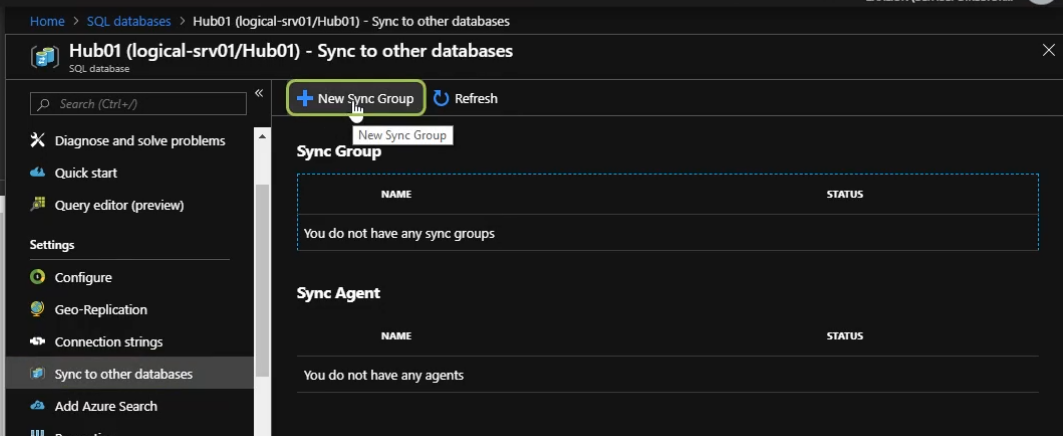
* SQL Data Sync uses triggers to update data between members. Since data sync is trigger-based, transactional consistency is not guaranteed. However, Microsoft guarantees that all changes are made eventually, and Data Sync doesn't cause any data loss.
* Data Sync users insert, update, and delete triggers to track changes. This means it also creates side tables in the user database for change tracking. So these change tracking activities have an impact on your database workload. So when using Data sync we might be interested in upgrading the access tier
* Other requirements:
  + Each table you want to use in Data Sync must have a primary key.
  + We shouldn't change the value of the primary key in any row. If we need to change the value of the primary key, simply delete that row and recreate it.
  + Snapshot isolation must be enabled.
  + A table cannot have an identity column that is not the primary key.
  + A table with the same name but different schemas are not supported.
  + Columns with user- defined data types are not supported.
  + Currently, Azure SQL Data Sync doesn't support Azure SQL Database managed instance.

**Demo: Creating hub and Member Databases**

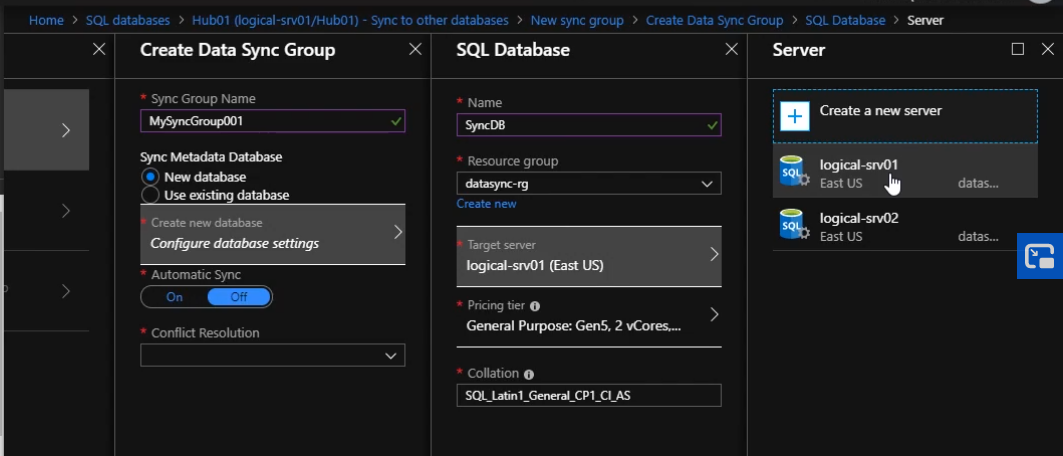
1. Create Hub database (as a single DB). Use Sample db during creation of the databases. Same Sample db will be created on member as well. This shall make sure both databases have same schemas to start with. Imp: The schema should first be same in both Dbs.
2. Create Member database.
3. Add client IP to firewall of both the DBs.
4. Connect to Hub database via SSMS and update few rows in Hub DB.

**Demo: Setting up Data Sync:**

1. To start configuring go to the HUB database dashboard and under settings click “Sync to other Databases”

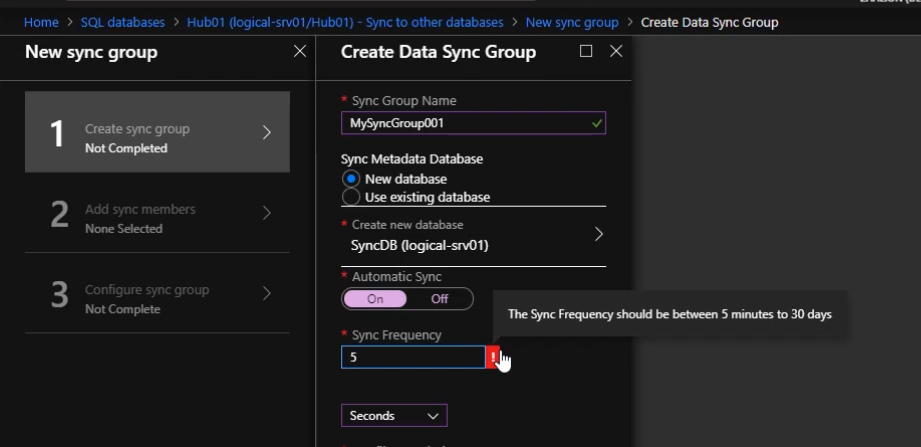


1. Here we add a new Sync group. Click on New Sync Group.

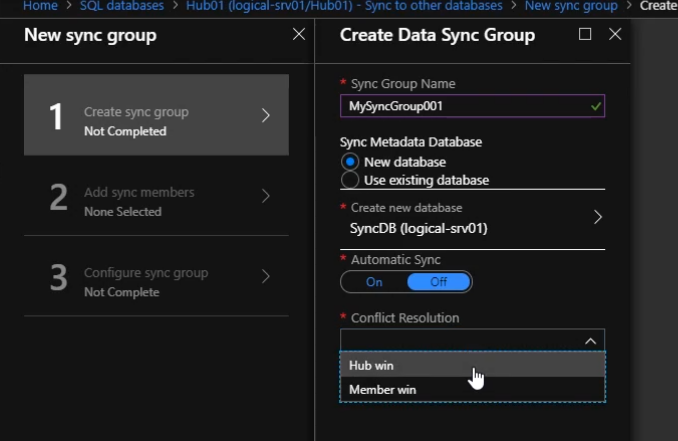


Notice that here we configure the Sync Database. This DB is created to maintain metadata. This DB needs to be a Azure SQL db and needs to be in same region as the Hub DB. We can put this sync db in the same server as the hub db, however note that this is not a requirement.

1. On the same page, we can schedule the sync. Note that Sync frequency should be between 5 mins and 30 days.



1. Next we configure the Conflict Resolution. We have two options : Hub Wins or Member wins.If you choose hub wins and a conflict happens between data in these two databases, Azure SQL Database is going to go with the hub version of the data



1. Next we add Member Databases to our Sync group

First we need to put the user credentials from my hub database. Data sync is going to use these credentials to connect to this database

