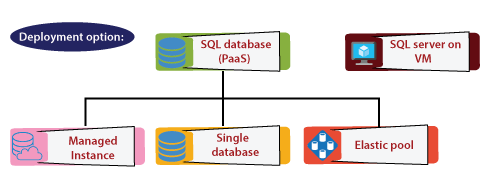
Azure SQL Database

SQL database is the flagship product of Microsoft in the database area. It is a general-purpose relational database that supports structures like relation data - JSON, spatial, and XML. The Azure platform fully manages every Azure SQL Database and guarantees no data loss and a high percentage of data availability. Azure automatically handles patching, backups, replication, failure detection, underlying potential hardware, software or network failure, deploying bug fixes, failovers, database upgrades, and other maintenance tasks.



There are three ways we can implement our SQL database

* **Managed Instance:** This is primarily targeted towards on-premises customers. In case, if we already have a SQL server instance in our on-premises data-center and you want to migrate that into Azure with minimum changes to our application and the maximum compatibility. Then new will go for the managed instance.
* **Single database:** We can deploy a single database on Azure its own set of resources managed via a logical server.
* **Elastic pool:** We can deploy a pool of databases with a shared set of resources managed via a logical server.

We can deploy the SQL database as an infrastructure as a service. That means we want to use the SQL server on an Azure virtual machine, but in that case, we are responsible for managing the SQL server on that particular Azure virtual machine.

Purchasing model

There are two ways we can purchase the SQL Server on Azure.

* **VCore purchasing model:** The vCore-based purchasing model enables us to independently scale compute and storage resources, match on-premises performance, and optimize price. It also allows us to choose a generation of hardware. It also allows us to use Azure Hybrid Benefit for SQL Server to gain cost savings. Best for the customer who values flexibility, control, and transparency.
* **DTU model:** It is based on a bundled measure on compute, storage, and IO resources. Sizes of the compute are expressed in terms of Database Transaction Units (DTUs) for single databases and elastic Database Transaction Units (eDTUs) for elastic pools. This model is best for customers who want simple, pre-configured resource options.

Azure SQL Database service tiers

* **General Purpose/ Standard model:** It is based on a separation of computing and storage service. This architectural model depends on the high availability and reliability of Azure Premium Storage that transparently copies database files and guarantees for zero data loss if underlying infrastructure failure happens.
* **Business Critical/ Premium service tier model:** It is based on a cluster of database engine processes. Both the SQL database engine process and underlying mdf/ldf files are placed on the same node with locally attached SSD storage providing low latency to our workload. High availability is implemented using technology similar to SQL Server Always On Availability Groups.
* **Hyperscale service tier model:** It is the newest service tier in the vCore-based purchasing model. This tier is a highly scalable storage and computes performance tier that leverages the Azure architecture to scale-out the storage and computes resources for an Azure SQL Database beyond the limits available for the General Purpose and Business Critical service tiers.

SQL database logical server

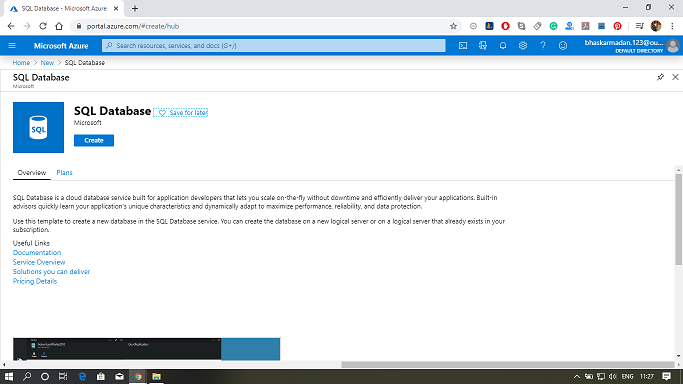
* It acts as a central administrative point for multiple single or pooled database logins, firewall rules, auditing rules, threat detection policies, and failover groups.
* It must exist before we can create the Azure SQL database. All databases on a server are created within the same region as the logical server.
* The SQL database service makes no guarantees regarding the location of the database in relation to their logical servers and exposes no instance-level access or features.
* An Azure database logical server is the parent resource for databases, elastic pools, and data warehouses.

Elastic pools

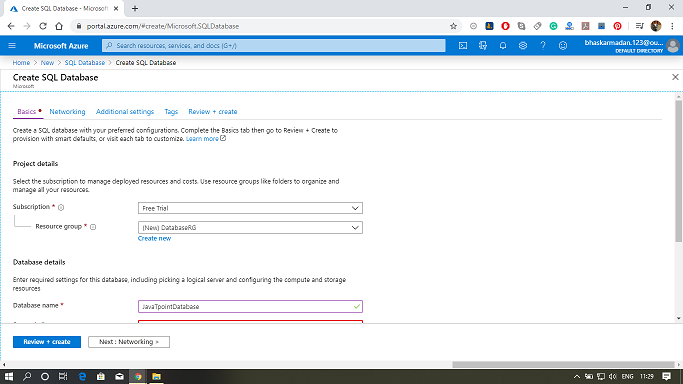
* It is a simple and cost-effective solution for scaling and managing more than one database. The databases inside an elastic pool are on a single Azure SQL Database server and share a group of resources at a fixed price.
* We can configure resources for the pool based either on the DTU- based purchasing model or the vCore-based purchasing model.
* The size of a pool always depends on the aggregate resource needed for all databases in the pool. It determines the following options:
  + The maximum resources utilized in the pool by the databases.
  + The maximum storage bytes utilized in the pool by the databases.

Creating an Azure SQL Database using Azure portal

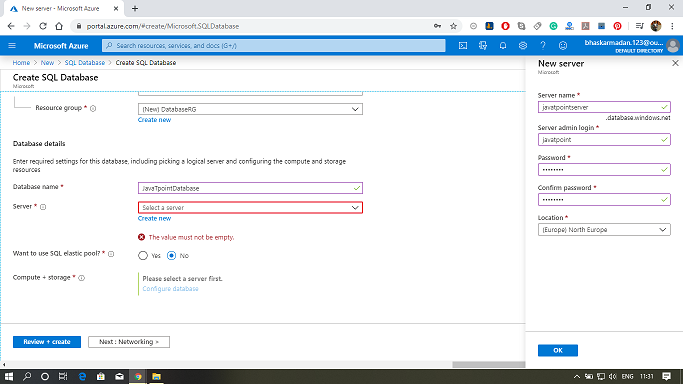
**Step 1:** Click on *create a resource* and search for SQL Database. Then click on create.



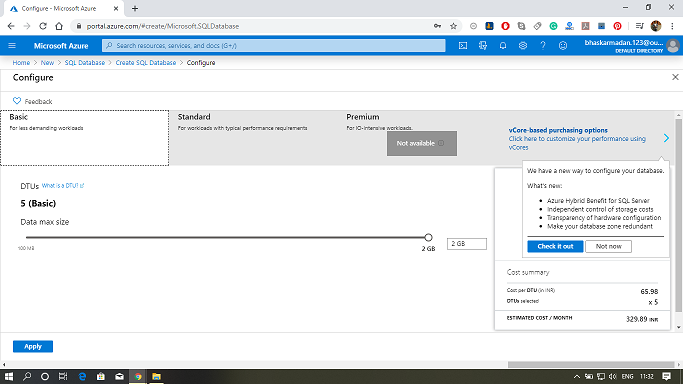
**Step 2:** Fill all the required details.



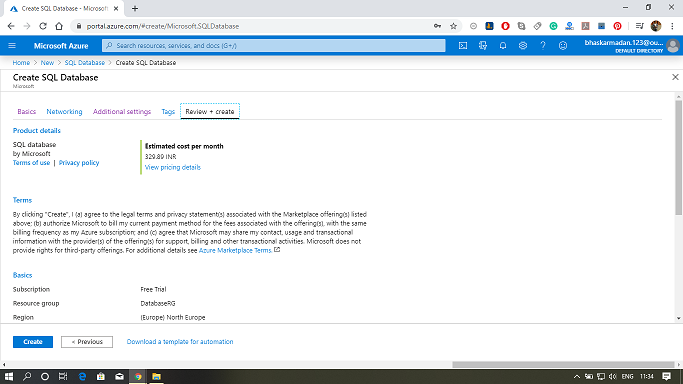
**Step 3:** Select a server or create a new one, as shown in the figure given below.



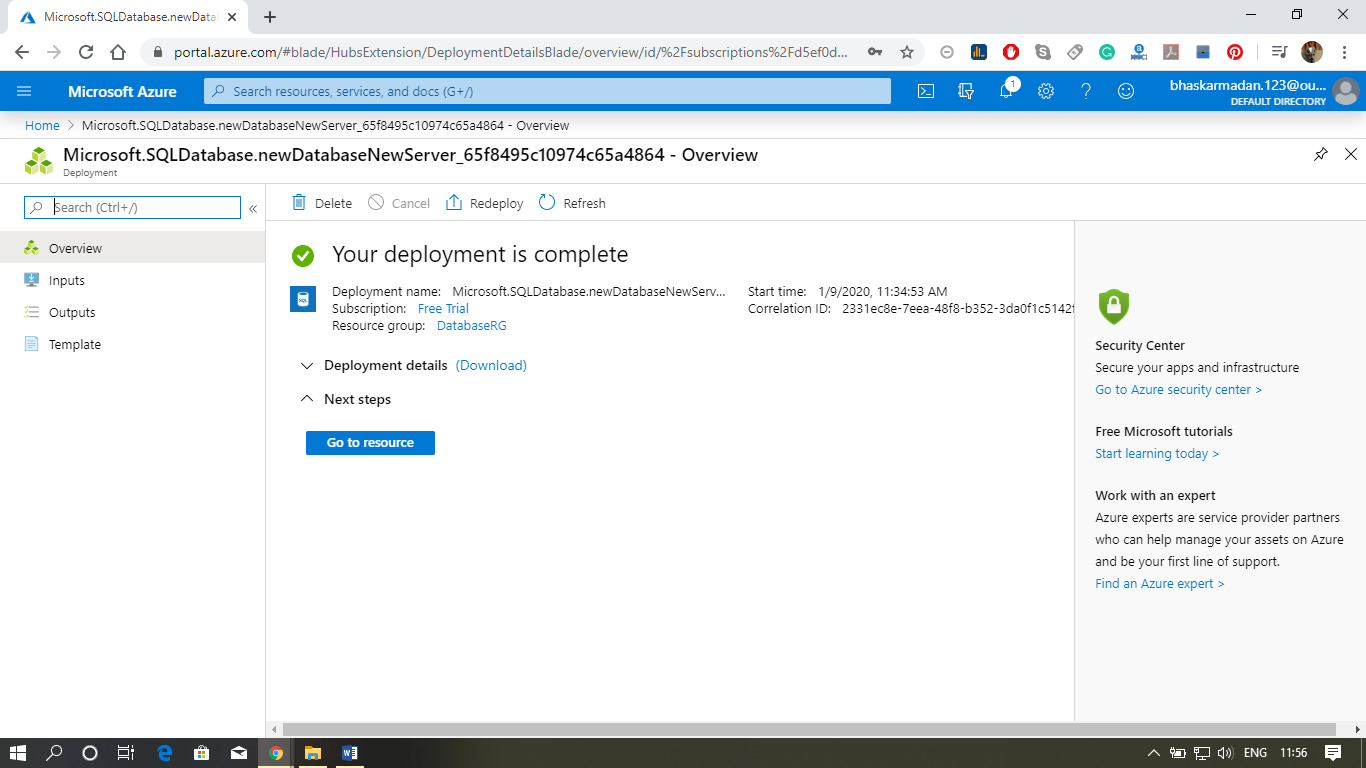
**Step 4:** Now, select the pricing tier by clicking on Compute + Storage, as shown in the figure below.



**Step 5:** After that, click on Review + Create and create the SQL database for your apps.



**Step 6:** Your SQL database is now created, now click on the go-to resources to configure additional settings for your database.



# Azure SQL Database Configuration

We'll see here the key configuration features of the Azure SQL Server and SQL database. In terms of Azure SQL database configuration, the first key thing is Firewall rules at a server level.

## **Firewall Rules**

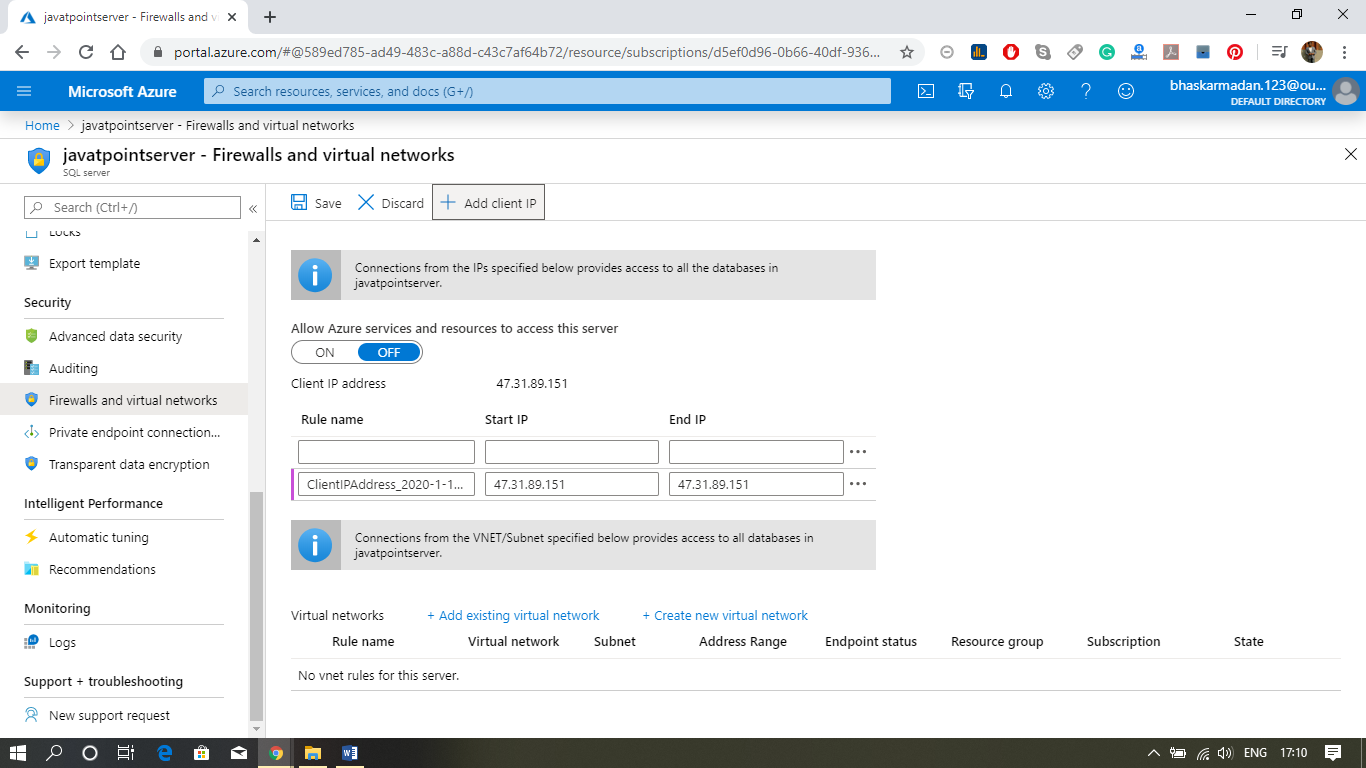
At a logical server within Azure, we can define some firewall rules. It can be IP rules. IP rules will grant access to the database based on the originating IP address of each request. And the second type of rule is the virtual network rule. It is based on virtual network service endpoints.

Rules for Azure SQL databases can be defined at two levels:

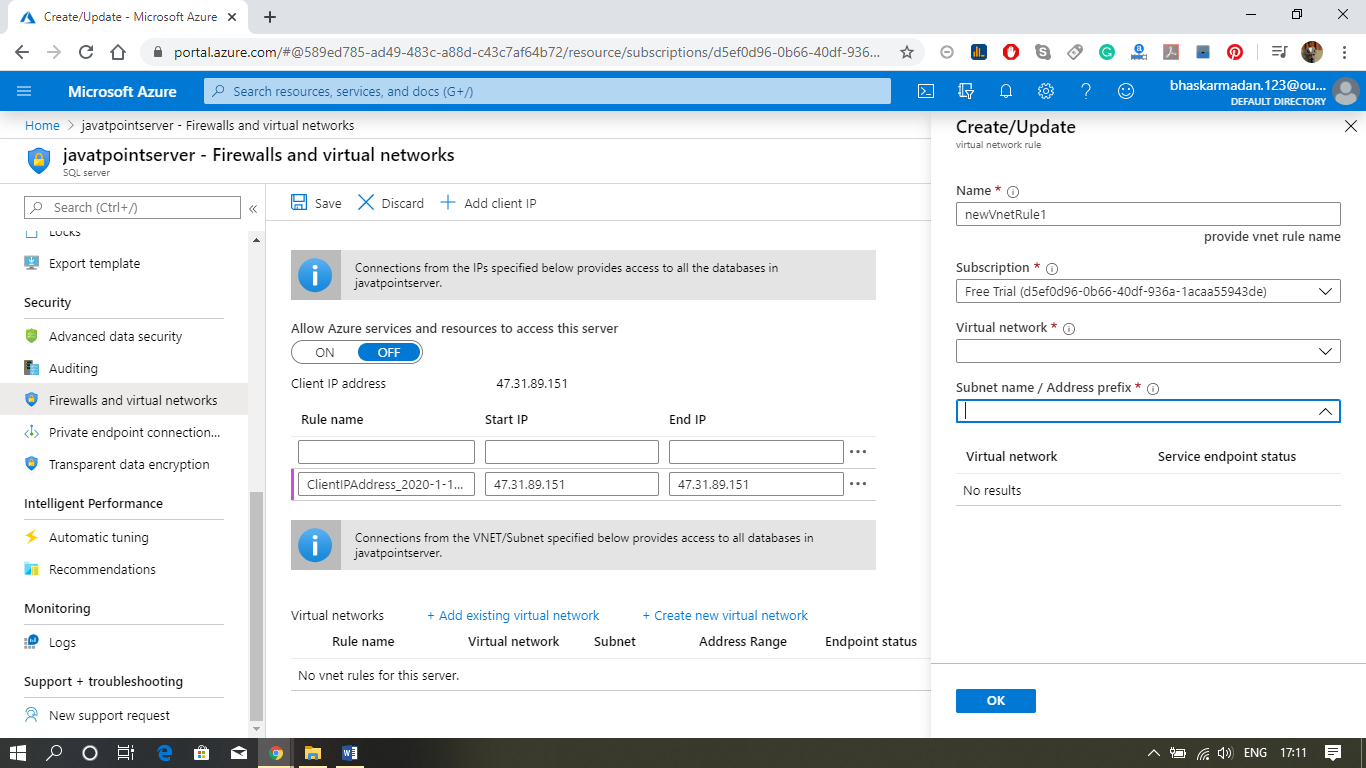
* **Server level firewall rules:** These firewall rules enable clients to access our entire Azure SQL server, i.e., each database within a similar logical server. These firewall rules will be stored in the master database. Server-level firewall rules can be configured by using the portal or by using Transact-SQL statements.
* **Database-level firewall rules:** These rules enable clients to access certain (secure) databases within the same logical server. We can create these rules for each database (including the master database), and they are stored in the individual databases.

### Configuring Firewall rules in Azure portal

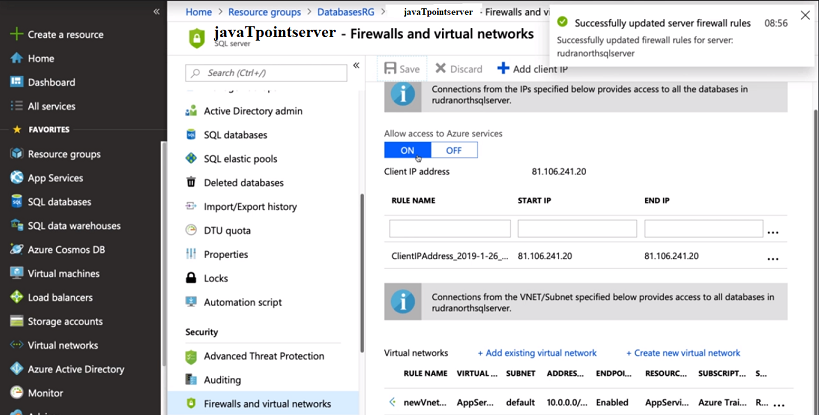
**Step 1:** Go to the firewall setting in your database server that you have already created. After that, click on ***Add Client***.



**Step 2:** Now, click on **Add existing virtual network** and fill the required details, as shown in the figure below.

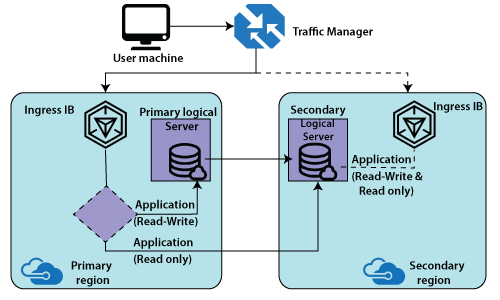


**Step 3:** Finally click on save, you will get the notification that your firewall rules got updated.



### Geo-Replication

It is defined at a database level, not server level, and it is designed as a business continuity solution that allows the application to perform quick disaster recovery of individual databases in case of a regional disaster or large scale outage.

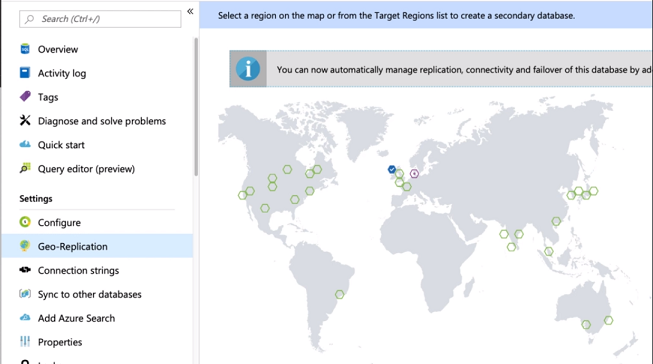


When we are configuring geo-replication, we specify a secondary database at a location far away from the primary location. We can have a traffic manager that routes the traffic by default to our primary load balancer and that the primary load balancer is based on the application request. If it is read and write, then it can route to a primary logical server. If it is 'read-only', it can route to a secondary server. Thereby the advantage of geo-replication is that we can offload some of the read-only traffic from primary and route to secondary.

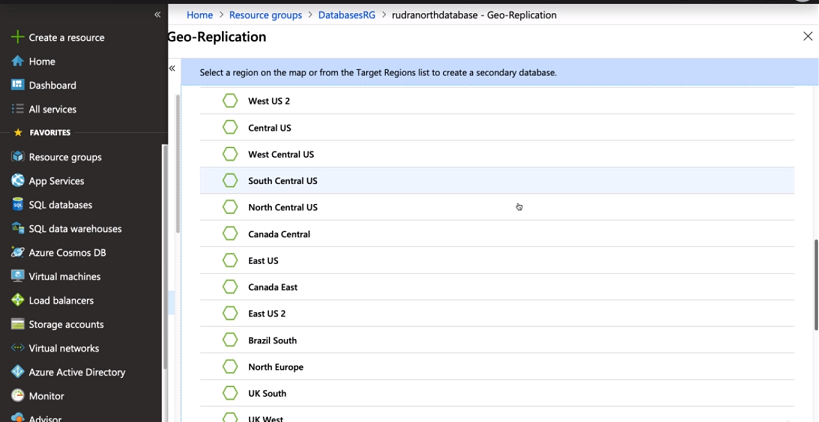
The primary performance will be good because read-only queries will consume a certain amount of CPU or DTU units installed that we have a secondary database where the data continuously get replicated.

### Configuring Geo-Replication using Azure portal

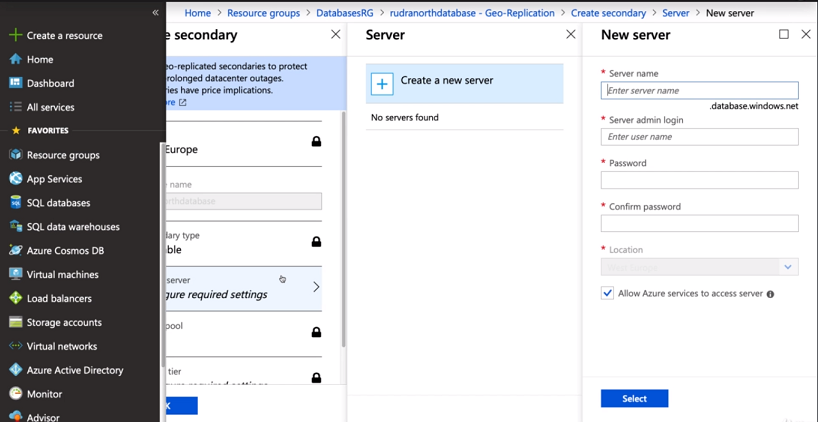
**Step 1:** Click on the Geo-Replication option; you will see the following window.



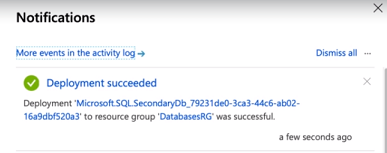
**Step 2:** Now, select the location where you want to replicate your database. You can choose multiple locations.



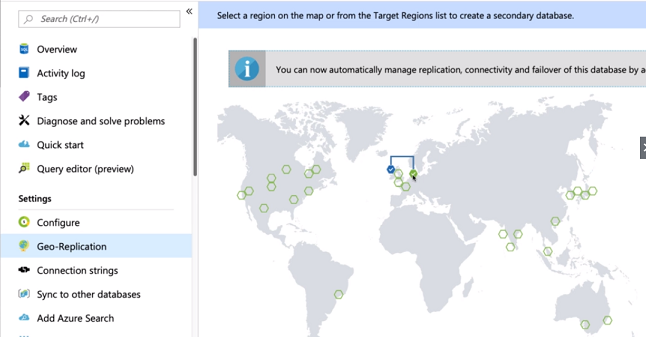
**Step 3:** Now, create a SQL server for the place where you want to replicate your data.



**Step 4:** Your server has been created and replicated successfully.



**Step 5:** You can see in the following figure, where the servers are replicated.



### Failover Groups

Auto-failover group is a feature of the SQL database that allows us to manage replication and failover of a group of databases on a logical server or all databases in a Managed Instance to another region.

We can initiate failover manually, or we can delegate it to SQL Database service based on a user-defined policy. When we are using auto-failover groups with automatic failover policy, any outage that impacts one or many of the databases in the group results in automatic failover. It allows the read-write SQL application to transparently reconnect to the primary database when the database change after failover.

### Database backups

The SQL database uses SQL server technology to create full, differential, and transaction log backups for Point-in-time Restore (PITR). The transaction log backups generally occur every 5-10 minutes, and differential backups occur typically every 12 hours, with the frequency based on the compute size and amount of database activity. Each SQL database has a default backup retention period between 7 and 35 days that depends on the purchasing model and service tier.

Long-term backup retention (LTR) leverages the full database backups that are automatically created to enable point-time restore. These backups are copied to different storage blobs if the LTR policy is configured. We can set an LTR policy for each SQL database and specify how frequently we need to copy the backups to the long-term storage blobs.