READIT STORAGE AND DB

SQL Database vs SQL Server :

1 : SQL DB is a PAAS Service which does not reside in a VNET but can be accessed via Service/Private Endpoints.

2 : SQL Server is an IAAS Service that is based on a VM and this instance of the Server already is in a Vnet by default.

- we use NSG's and enable VPN Gateway and/or Bastion

I : Creating the DB in Azure and then Connecting to it using VS Code :

For the Application Purpose, we will create a basic Sql DB with below Configurations :

- Authentication Method : Sql Authentication (for now)
    - Will usually use Sql and Entra Auth method

- No Elastic Pool
- Basic Service Tier
- Ensure TLS version is the latest one (>=1.2)

<u>After the DB is successfully created, we should connect to it from VS Code using the following steps :</u>

1 : Install SQL Server extension.

2 : Under Add Connections, select it and then select the Connection String from **Connection Strings** in Sql DB in Azure.

3 : Paste the connection string in the VS Code top bar and then add the pw.

4 : Add a name to the Db that will be listed in the VS Code extension.

5 : Inorder to access the DB from VS Code, we need to set the firewall rules on Sql Server as the multiple DB's are hosted on a single server and firewall rules apply to server level.

6 : If accessing from our local system, add the **public IP** of our local machine to the Sql Server using its Networking config.

- Select **Selected Networks** and add the IP.

7 : Add the Connection String again to the Vs Code extension as before and save and then try connecting to the database. It should be listed in the extension if successful.

II : Connecting the Catalog VM to the Sql DB.

1 : We need to make couple of changes first :

- in **Startup.cs** file, change the *useInMemory* variable to false
- in **appsettings.json,** add the connection string to the *ConnectionStrings* attribute *BooksDB* key from the Azure Sql DB.

2 : Download a tool with the following command to enable entity framework in working with our DB.

- *dotnet tool install --global dotnet-ef --version 8.0.10*
    - This command installs the Entity Framework Core (EF Core) CLI (Command Line Interface) tool globally on your machine.

3 : The following command will create a migration files for the database.

- *dotnet ef migrations add -c BookContext InitialCreate*

4 : The following command will create a connection with the DB and create and update a tables inside it.

- *dotnet ef database update*

5 : We will see the updated tables in the SQL extension. It will be empty.

6 : Inorder to populate the table with data, we will have to run the application and then Load the books after running the application.

7 : Again try running the DB and the books will be populated.

We ran the application locally now. The DB save the application's data as we ran the application with the DB string.

Now we have to run the application in the cloud in the Catalog VM and connect the VM to the SQL DB.
The steps are as following :

1 : We will start by redeploying the catalog application. Publish the application by running the publish command in VS Code as we have made changes to the application code. Copy the contents of the publish folder which has the new connection string and other changes to the catalog VM's Catalog directory in C drive.

2 : Now we will have to establish the connection between the VM and the DB. We cant connect to the DB directly. Instead we need to run the catalog application using the Gateway's public IP as that is how we have our implementation of the GW.

3 : The application will run with the GW public IP but it wont display any books previously populated as it wont be able to

connect to the SQL server as catalog application is not allowed to access the DB Server.

So we will add the VM's public IP to the DB Server's firewall so that the Catalog application can access the DB. This is not a secure way to establish a connection.

4 : Inorder to secure the connection, we can create a private endpoint in the SQL Server's Networking settings which will allocate a private IP to the Sql Server from within the Vnet we specify housing the Catalog VM.

Implementing a private endpoint will change the how we connect to the Sql Server as the DNS record will change as we will connect to the Pvt IP of the SQL Server. This is worth considering as we are connecting to the SQL server from our code (appsettings.json file) using the SQL server name.

Now we can essentially delete any allowed IP's from the SQL Server's firewall setting except for our local machine's IP as we are using it to access the SQL Server from our application.

5 : We can now delete the VM's Public IP from the DB Server's firewall as we have now established a private connection. We can still run the Catalog VM with the GW PIP which will still have the access to the SQL Server as they are virtually in the same Vnet because of the Private Endpoint.

III : Connecting the Inventory App Service to the Sql DB.

We will connect the Inventory App Service to the Sql DB Server by creating a Vnet Integration.

The steps are as follows :

1 : We need to make changes to the **index.cshtml.cs** file in the inventory service. In that file, we need to uncomment **OnGet** method.

2 : We should then make changes to the connection string in appsettings.json and add the sql db connection string and then deploy the application on Web App using VS Code.

3 : If we do not make changes to the connection string, we would get an error. We will still be able to access the Web App on AG's Public IP as we have configured it already but the Web App wont be able to access the Sql Db as we have not whitelisted the IP.

4 : Now inorder for the Inventory App Service to connect to the SQL Server, we will need to configure Vnet Integration (enables outbound connections from PAAS Resource to Vnet) on the App Service which will help it access the DB securely.

5 : From the Configuration Blade of the App Service, add the Connections Strings which would restart the Web App.

6 : *Ideally, we need to configure both ie. Vnet Integration (Outbound traffic from App Service) & Private Endpoint*

(Inbound Traffic into the Vnet from App Service) which is a safer approach.

7 : But in this case, we already have a Private Endpoint configured in the Default Readit-app Vnet, so the all of the resources within the Vnet can communicate with the App Service and not only the resources from the subnet we configured the Private Endpoint with.

The App Service by default uses https but if we connect to a VM deployed in a Vnet using *Vnet Integration*, the App Service might default using http if the VM is not using https.