# Azure Training

8th July: PostgreSQL on Azure VM

Steps:

1. Create a virtual machine in Azure dashboard
2. Copy its IP address
3. Install Microsoft Remote Desktop
4. Login using our created VM’s IP
5. Install PostgreSQL
6. In conf file: pg\_hba : host all all <your-mac-ip>/32 md5
7. In conf file: postgresql.conf: listen\_addresses = '\*'
8. Create an inbound rule in Azure dashboard with port number 5432, tcp protocol, and allow actions - this rule enables/maintains traffic from azure to VM. -> facilitates communication from azure to VM – allows to send packets.
9. Create an inbound rule windows firewall of VM (Win+R => wf.msc) with same details – this rule enables/maintains process of packets from azure to VM. -> facilitates acceptancs of any communication from azure to VM – allows receiving of packets. (BOTH RULES MUST BE PRESENT)
10. Restart PSQL – stop and start again in cmd
11. Now in Mac terminal check connection: nc -zv <vm-ip> 5432 # should say “succeeded”
12. Also try to open command terminal : psql -h <vm-ip> -U postgres -d <your\_db\_name>
13. Now change your connection string in webapi project and migrate again.
14. Check for database existence in VM PostgreSQL admin.

09th July: Deploy Dotnet API on Azure VM

Steps:

1. Create VM: Image (OS) - > Ubuntu 20.04 x64 Gen1; Enable ports: 80 (HTTP), 22.
2. At the end of creating VM download the key value pair as pem file for later use.
3. As we are using Ubuntu, we will not have RDP in our inbound rules, so we cannot open VM using Microsoft remote desktop by just giving the IP.
4. Open terminal in mac: ssh -I “path to your .pem file” azureuser@IP
5. Now after entering your VM: Install, enable and start docker. Check for docker version for clarifying.
6. Now that your docker is ready to action, get your web Api ready, just start a new web api project at 9.0 – creates a basic weather api.
7. Add docker file: stages: 1. Compile: ASPNetcore 9.0 sdk as build, copy all files, and run publish command(complies), 2. Runtime: ASPNetcore 9.0 as aspnet as runtime, copy all published files from build, expose and give entry point.
8. After docker file creation, build – and push into docker hub.
9. Now go back into VM and pull the image.
10. Containerize it by run command (daemon off, port mapping).
11. Now that the container runs, so does the application, check any endpoints with curl command, remember we are using VM so instead of localhost, enter your VM IP.
12. Here curl http:IP:/weatherforecast - and it returns the value.

ARM -> Templates for creating server and DB in azure - > deploy.json and paramters.json – give appropriate locations, server name and password. And deploy using command and see for existing server and db under the used resource group in azure dashboard.

10th July: Azure Board:

Epics, user story(agile mode), tasks etc…

11th July:

Blob storage – containers.

Task: Create a container and connect it with a web api backend and perform file upload and download

Steps:

1. Create a new web api project, and appropriate services and controllers for file upload and download functionalities.(dto and stuffs)
2. In app settings.json add your link to your container / blob’s connection string (security -> access key) and the appropriate container name.
3. Make a connection in service, by aptly configuring it.
4. Refer Blob service client (get the container and get to upload the filestream), blob container client (directly upload the file stream) – Gayathri’s Git.
5. Swagger – upload file and check in container if it exists. And download and see.

Task: Static website: FE in storage

Steps

1. Blob – data management – static website – enable – default, error: index.html
2. A container is created named $web in the blob and will be searching for default(index.html as given) to execute and show in its primary endpoint(link)
3. Now in your angular app: ng build --configuration production
4. This build a dist file – browser – index.html; main.js; polyfills.js; styles.css
5. Now in your $web: upload all the four files above
6. Now you will see your angular app in the primary endpoint of you blob (but backend connectivity will damage because of CORS)

Task: Upload project files and log

Steps:

1. VS code – extensions – azure storage – install.
2. Sign in and look onto your favored container – right click – upload folder – select your project.
3. Log: either customize serilog in program.cs
4. Or perform upload operation in task1 but with Log files….and make it automated by using Start async (as it will execute at the start of runtime of the application)
5. LogsUploadService -> starts async -> searches for Logs folder (serilog) and uploads todays file into the specified container (logsfile)